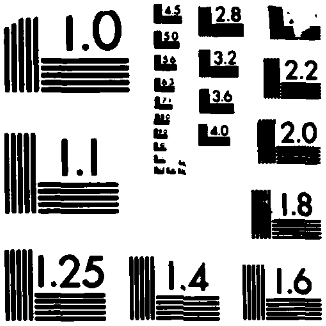
AD-A138 309 ROBUST MULTIVARIABLE CONTROLLER DESIGN VIA IMPLICIT 1/4
MODEL-FOLLOWING METHODS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W G MILLER
UNCLASSIFIED DEC 83 AFIT/GE/EE/83D-48 F/G 14/2 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

ROBUST MULTIVARIABLE CONTROLLER DESIGN VIA IMPLICIT MODEL-FOLLOWING METHODS

THESIS

AFIT/GE/EE/83D-48

William G. Miller
Capt USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/83D-48

ROBUST MULTIVARIABLE CONTROLLER DESIGN

VIA IMPLICIT MODEL-FOLLOWING METHODS

THESIS

AFIT/GE/EE/83D-48 William G. Miller
Capt USAF

AFIT/GE/EE/83D-48

ROBUST MULTIVARIABLE CONTROLLER DESIGN VIA IMPLICIT MODEL-FOLLOWING METHODS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

William G. Miller, B.S., M.S.S.M
Capt USAF

Graduate Electrical Engineering

December 1983

Approved for public release; distribution unlimited.

Preface

At the end of the week prior to the planned final typing of this thesis, a programming error was discovered by the author of the software which produced the controller designs upon which much of the report is based. Investigation revealed that the design software error had an obvious, significant impact on some of the results, observations and conclusions presented. Considering the limited time available, the most reasonable option appeared to be to document the error, and to explain its effects on the results of this study as fully as possible. Thus, except for the addition of Appendix E and the references to it in Chapters IV, V and VI, this report has been left much as it was before the error was found. The theories and formulations presented are still valid, as are the logic, tools and methodology used to pursue the design and analysis of the controllers. The controller designs developed are valid and do possess solid capabilities despite the software error; they were achievable due to the iterative nature of modern optimal control design techniques. These iterations yielded controllers with gains that produced desirable performance; in using the corrected software, different weighting matrices would be chosen (generally, as the result of easier design iterations) to produce essentially the same gains and the same closed-loop system performance.

While I am disappointed that the accuracy of the data and some of the observations presented in this report has diminished, I am glad that the discovery of the error occurred early enough to allow it to be identified and its effects addressed, albeit in a limited sense. I sincerely hope that those who may read this thesis will still find it to contain useful information.

I wish to thank my thesis advisor, Dr. Peter S. Maybeck, for helping to make this project such a valuable learning experience. His expertise and guidance contributed a great deal to my work and to the final form of this report. I would also like to (personally) thank him for his patience with my tendency to (always) split infinitives. Thanks are also due Dr. Robert A. Calico, Jr. for his help with modelling issues and his review of the manuscript for this report.

I was also very fortunate to have the assistance of two "unofficial" thesis committee members -- Mr. Finley Barfield and Capt Rich Floyd. I would like to express my sincere appreciation for their interest in this study and for the time they spent helping me to grasp the significance of many of the analysis and design issues that arose in the course of my work.

Last, but certainly not least, I would like to thank my wife, Sue, for her constant encouragement and patient understanding throughout the course of this effort.

William G. Miller

Contents

Contents

List of Figures

List of Tables

Abstract

This study applies the concept of implicit model-following to the design of a multivariable control system based on the Linear system model, Quadratic cost, and Gaussian noise process (LQG) assumptions of optimal control. The design objective is to achieve improved controller robustness in the face of "uncertainties" arising from the differences between the simplified linear model used for controller design, and a more realistic truth model representation which includes higher-order dynamics, parameter variations, and nonlinearities.

This report introduces the concept of implicit model-following and reviews the formulation of its incorporation into the design of a controller that consists of a Command Generator Tracker (CGT) employing a Proportional-plus-Integral (PI) inner-loop regulator and, if required, a Kalman Filter (KF) to provide state estimates for feedback control (CGT/PI/KF controller).

The theoretical background for the application of matrix singular value analysis to the study of controller robustness is presented, and a computer program is developed to apply this type of analysis to the general CGT/PI/KF controller configuration. Additional software is developed to analyze the performance of the specific designs achieved in this study by means of conventional simulation with respect to various realistic truth models. Both programs are documented and listed in this report.

A variety of deterministic CGT/PI controllers for the decoupled, pitch-pointing control of a modern fighter aircraft (the AFTI F-16) are designed through the use of software developed in the course of previous AFIT thesis efforts. These controller designs are analyzed extensively through the use of the software developed in this study. The incorporation of implicit model-following in the design process for the inner-loop regulator is shown to provide an enhanced capability to produce designs which work well despite unmodelled dynamics and system nonlinearities (specifically, control actuator saturation), when compared to design methods which do not use implicit models.

The results of the singular value and simulation analyses are combined and contrasted. The type of singular value analysis employed in this study is shown, for the design problem considered, to be of limited value in predicting the robustness of controller performance. Classical analysis based on the locations of nominal closed-loop controller poles is shown to provide considerable insight and predictive capability for robustness in the face of higher-order dynamics.

ROBUST MULTIVARIABLE CONTROLLER DESIGN
VIA IMPLICIT MODEL-FOLLOWING METHODS

I. Introduction

1.1 Background

An important objective in control system design is to meet constraints on both the transient and the steady-state behavior of a physical system. Achieving these objectives generally requires the use of some form of feedback control, which provides the designer with an even more critical concern -- that the closed-loop system be stable at all times and under all operational conditions. Design of systems that meet these criteria is greatly complicated by unknown or unmodelled differences between the mathematical model on the drawing board and the real world. These include (1) disturbances that affect the controlled system (plant) and sensors, (2) order reduction in the design model required to simplify the design or controller implementation, (3) unmodelled or ignored nonlinearities, and (4) parameter variations between the actual system and an otherwise adequate design model representation. The effects of these "uncertainties" can cause serious performance degradation or instability if the designer does not allow for their existence. The term "robustness" is used to define the amount of design uncertainty that can be tolerated, or the amount by which the real world can vary from the nominal design model, without destabilizing the actual physical closed-loop control system or causing it to fail to meet performance specifications [4,11,13,30,41]. In this thesis "robustness" will be primarily used in reference to the stability consideration; when "performance robustness" is intended, it will be specified explicitly.

The stability robustness of single-input, single-output (SISO) control system designs is frequently characterized using the well known concepts of gain and phase margins, although these measures do not take into account the effects of simultaneous gain and phase changes that could produce instability [30,43]. Even these limited measures have no direct counterparts that can be applied to the case of multiple-input, multiple-output (MIMO) control systems. Attempts to analyze the robustness of MIMO designs one loop at a time using the SISO analysis methods generally provide overly optimistic robustness estimates in that they fail to identify the effects of simultaneous variations in interrelated system loops [11,12,14,41]. Since many of the flight control and weapons system developments currently of interest to the Air Force depend on multi-loop designs, the lack of adequate design and analysis tools has a serious impact. A means of quantifying the robustness of MIMO designs is a necessary prerequisite for developing more certain methods of synthesizing control systems that are, in fact, robust. Singular value analysis of the matrices which are used to describe the control system has been shown to provide such a capability [4,7,11,12,14,15,18,24,27,30,35,37,38,41], and it will be described in some detail and used in this thesis.

One approach to modern multi-loop control system design involves the assumption of a linear system with disturbances and measurement corruptions which can be represented as Gaussian processes, and with control laws based on the mathematical optimization of a scalar quadratic cost function (performance index). Such LQG (Linear-Quadratic-Gaussian) design techniques can be applied to many types of complex problems, generating designs with many desirable characteristics. These techniques do not, however, directly address the robustness issue, and many designs based on LQG assumptions and methods have demonstrated a serious lack of robustness; the presence of an observer or filter in the control loop is a primary cause of the problem [10,30]. Robustness analysis is therefore imperative for the LQG designer.

One controller configuration in the LQG class consists of a Command Generator Tracker (CGT) which provides feedforward control, a Proportional-plus-Integral (PI) inner-loop controller designed by LQ deterministic optimal control methods, and a Kalman Filter (KF) to provide state estimates necesssary for control law generation (as a unit, henceforth referred to as CGT/PI/KF) [16,31,34]. The CGT/PI/KF controller was the subject of two recent AFIT thesis efforts by Capt R. M. Floyd and Lt A. Moseley [16,34]. These studies included the development of interactive computer software for efficient design and performance analysis of the CGT/PI/KF controller. A unique aspect of this type of controller is the use of "model-following" to force the controlled plant to respond in a way which mimics the response of an ideal system. This can be done by means of the feedforward controls of the CGT ("explicit" model-following), or by changing the control system's closed-loop characteristics by incorporating a model of an ideal response into the regulator's performance index ("implicit" model-following), or both. The complexity of this configuration makes robustness analysis somewhat difficult. At the same time, the degrees of design freedom created by this compexity make the CGT/PI/KF a promising vehicle for investigating means of robustess enhancement, especially through the use of implicit model-following. Refer to Chapter II for a more detailed discussion of model-following control and, in particular, the structure and design methods for the CGT/PI/KF controller.

1.2 Recent Research

In general, the scalar parameters that describe a SISO control system are replaced by matrix quantities in the MIMO case. The open-loop transfer function becomes a loop gain matrix, and the return difference function (the denominator of the control ratio) is replaced by a corresponding return difference matrix function. Analysis of functions of these matrix quantities can provide much insight into the performance and stability robustness of complicated systems, just as scalar analysis does in the SISO case [11,14,30,37,41]. Many of the classical "rules of thumb" that have traditionally been applied to the loop gains of SISO control systems can be generalized to apply to the "singular values" of the MIMO loop matrix [14,37,43]. Similarly, the use of the matrix singular value decomposition provides insight into the "size" of the system's return difference matrix; this quantity and the associated inverse return difference matrix are directly related to the system's overall stability [11,14,30,47,43].

Matrix singular value analysis (discussed at length in Chapter III) has generated a great deal of research in the last few years [4,7,11,12,14,15,18,24,27,30,35,37,38,41]. A very good basic discussion of the impact and application of singular value analysis in assessing stability robustness was presented by Doyle [11]. Expansion of these concepts to the more general analysis of both stability robustness and performance using singular value analysis of several system matrices was given by Doyle and Stein [14]; this work provided some pleasing generalizations of many classical SISO design tools to the MIMO case. In both of these sources, the robustness analyses are based on the effects of a very general class of perturbations to the design model parameters referred to as "unstructured" uncertainties. Since a subset of this class generally represents perturbations that could never physically occur, robustness estimates are often overly conservative. More recent research [12,18,27,40] has centered on analysis that takes into account knowledge of what perturbations are physically possible and which will have the greatest effect on robustness and performance. This work with "structured" uncertainty leads to tighter bounds on robustness estimates.

Robustness analysis is necessary to ensure that a control system will remain stable in the face of uncertainties. If such analysis uncovers design deficiencies, then steps must be taken to robustify the system. Gilbert [19] has shown that if the eigenvectors of the closed-loop system are made as nearly orthogonal as possible, improvement in robustness results; design methods involving eigenstructure assignment therefore have the potential for robustness enhancement [1,20,25]. Because it depends on an internal model of the overall system to provide estimates, insertion of a Kalman filter in the loop generally degrades robustness [10,30]. Doyle and Stein [13] have shown a means of recovering some of this lost robustness by systematically injecting pseudonoise into the filter model at the control input points; a dual approach involves a structured change in the quadratic state weightings used in the controller design [14,26,43]. The first of these methods was employed in an AFIT thesis by Capt E. D. Lloyd [28] with some success. Except for the methods cited, much of the design robustification that occurs must still be based on iterative methods, using insight gained from such tools as singular value analysis and simulation of actual system performance.

1.3 Objective

The purpose of this study was to attempt to develop synthesis techniques for the design of robust LQG controllers. In particular, an effort was to be made to develop the use of implicit model-following as a means of enhancing controller robustness in the face of a variety of uncertainties. The software developed by Floyd and Moseley was to be used to conduct the design and initial evaluation of CGT/PI controllers. A computer program was to be developed to calculate and plot the singular values of the loop gain and inverse return difference matrices of a control system in the CGT/PI/KF configuration, as a function of frequency. An inequality (discussed in Chapter III) relating the singular values of the inverse return difference matrix and the maximum stable perturbation to the design model was to be used as one measure of stability robustness. Additional software was to be written to conduct performance analysis of the controllers operating on nonlinear truth models, models with higher-order dynamics, models with parameter variations, and combinations of such perturbations to the design model. The results of singular value and simulation analyses were to be compared and combined. Methods of improving the robustness of the control system, primarily with implicit model-following, without excessively or unnecessarily degrading performance were to be investigated.

1.4 Sequence of Presentation

Chapter II presents a more detailed look at the structure and design of the CGT/PI/KF controller, as well as the concept of model-following. Chapter III outlines the development of the robustness analysis and enhancement methods that were to be employed. Chapter IV presents specific design models and objectives, the approach to be taken in conducting the controller design and analysis, and a summary of general observations that emerged during the study concerning the use of implicit model-following in regulator design. Chapter V discusses the specific results of the design efforts conducted, and Chapter VI offers some conclusions and recommendations for further study. Appendix A contains a brief review of the LQG problem formulation and solution, while Appendices B, C and D document the software developed and/or used in this study. Appendix E documents an error that was discovered in the CGT/PI/KF design software after the design work of this research had been completed. The error and its significance with respect to the results and conclusions presented herein are outlined, and some preliminary designs using the corrected software are presented. The reader is urged to review Sections E.1 through E.3 prior to any detailed study of Chapters IV, V or VI.

1.5 Notation

An attempt has been made to adhere to the notation used by Maybeck [29-31] and the AFIT theses by Floyd and Moseley [16,34]. Appendix A contains a brief review of the LQG controller problem formulation and solution; this includes an introduction to the notation as well as the concepts represented. This review includes the majority of the basic notation that is used throughout the thesis, and additional notation is defined as it is introduced.

II. Model-Following and The CGT/PI/KF Controller

2.1 Introduction

This chapter discusses the concept of model-following control in LQG designs. The CGT/PI/KF controller configuration is described [5,30], along with an outline of a design procedure as implemented in software developed by Floyd and Moseley [16,34]. The level of detail in this description is intended to be sufficient for the reader interested primarily in understanding the purpose and results of the study described in this thesis. While much of the theory discussed is applicable to a wide range of problems, the analysis software developed in this study is useful only for designs based on the formulations of, and conducted using the software of, the previous thesis efforts. For that reason, a reader interested in using the specific tools developed herein is referred to the Floyd and Moseley works for a more complete development and description of related design software and methodology.

2.2 Model-Following Controllers

The development in Appendix A demonstrates that the LQG design approach provides a systematic means of synthesizing control laws for complex, multi-loop systems. Despite the capabilities inherent in methods based on optimal estimation and control theory, classical control design and analysis methods remain the primary tools of most design engineers [16,31,43]. One of the reasons for the failure of modern tools to replace the classical is the fact that many design specifications are stated in terms of time-domain input/output behavior -- settling time, peak overshoot, damping ratio, steady-state error and disturbance rejection characteristics -- which are not readily translated into quadratic weightings to be used in a performance index. The result is a considerable amount of trial-and-error-based design iteration, often without a great deal of physical insight.

The concept of model-following first arose in the 1960's as a means of implementing modern control techniques, and has proven particularly useful in aircraft control system designs where specifications consist of desired "handling qualities" expressed in terms of classical time-domain criteria [16]. Model-following provides a means of forcing the outputs of a controlled system to behave as if they were those of a model system which is known to have the desired qualities. Typically, the objective is to cause a complex, high-order system to match the characteristics of a simpler model, achieving an approximation of a first- or second-order output response that meets specifications. Another use of model-following is in causing one system to mimic the response characteristics of another system, such as one aircraft that exhibits the same handling qualities as another [16].

Within the model-following class of controllers are two subgroups: implicit and explicit model-following controllers. In implicit model-following, the description of the "model" system is incorporated into the performance index; feedback control is thus designed to penalize the system mathematically for deviations from model performance. In explicit model-following, the desired response dynamics are simulated in the controller and used to generate commands consisting of feedforward gains on model controls and states. The feedforward control, generally used in conjunction with feedback control from some form of inner-loop regulator, thus optimally drives the system to achieve model response. Development of both concepts is quite straightforward [6,16,34].

A very simple example formulation follows, which demonstrates how model-following works. The fact that this is an example formulation cannot be overemphasized. There are similarities between the controllers discussed in this section and those actually used in this study, but there are important differences as well. To avoid confusion that could result from incorrectly relating this simple example formulation to that of the PI regulator developed in Appendix A and the CGT/PI development of Section 2.3, distinct notation is used for the weighting matrices of the cost functions in this section. Reference to the equations of this section is avoided throughout the rest of the thesis, except in Appendix A, where the correct relationship between the different formulations is discussed.

Consider the linear-quadratic full-state feedback (LQSF) regulation problem for the continuous-time system described by the state differential equation

$$\dot{\underline{x}}(t) = \underline{A}\underline{x}(t) + \underline{B}\underline{u}(t) \qquad \text{(II-1)}$$

While it is possible to include non-zero command inputs in the formulation, the result would be much more complicated; in this simple example, only response to initial conditions is considered. The desired control law is of the form

$$\underline{u}(t) = - \underline{G}_c(t)\underline{x}(t) \qquad \text{(II-2)}$$

A steady-state, constant gain, $\underline{G}_c$, is often sought for practical implementation. The objective is to force the output of the system

$$\underline{y}(t) = \underline{Cx}(t) \tag{II-3}$$

to match as closely as possible the output of a model system, described by the equation

$$\dot{\underline{y}}_m(t) = \underline{A}_m\underline{y}_m(t) \tag{II-4}$$

where $\underline{y}_m$ is of the same dimension as $\underline{y}$. At this point, the use of time arguments will be dropped to simplify the notation. The appropriate scalar performance index to be minimized in determining the optimal feedback control law is

$$J_I = 1/2\int_0^\infty [(\dot{\underline{y}} - \underline{A}_m\underline{y})^T\underline{Y}_I(\dot{\underline{y}} - \underline{A}_m\underline{y}) + \underline{u}^T\underline{U}_I\underline{u}]dt \tag{II-5a}$$

$$= 1/2\int_0^\infty [\underline{x}^T\hat{\underline{Y}}_I\underline{x} + 2\underline{u}^T\hat{\underline{S}}_I\underline{x} + \underline{u}^T\hat{\underline{U}}_I\underline{u}]dt \tag{II-5b}$$

where

$$\hat{\underline{Y}}_I = (\underline{CA} - \underline{A}_m\underline{C})^T\underline{Y}_I(\underline{CA} - \underline{A}_m\underline{C}) \tag{II-6a}$$

$$\hat{\underline{S}}_I = \underline{B}^T\underline{C}^T\underline{Y}_I(\underline{CA} - \underline{A}_m\underline{C}) \tag{II-6b}$$

$$\hat{\underline{U}}_I = \underline{U}_I + \underline{B}^T\underline{C}^T\underline{Y}_I\underline{CB} \tag{II-6c}$$

and the function is integrated over all time so as to define a steady-state, constant-gain controller.

The cross-weighting term, $\hat{\underline{S}}_I$, appears in the index due to the need to track rates of change in the output. The form of the performance index shown thus allows the use of standard LQ synthesis techniques to develop an optimal feedback controller that tries to force the system to meet classical performance specifications embodied in the model output, $\underline{y}_m$. The extent to which the desired response is achievable is dependent on the degree of difference between the system's inherent dynamics, represented by $\underline{A}$, and the desired response dynamics, represented by $\underline{A}_m$.

In developing, again as a very simple example formulation, an explicit model-following controller for the same system, an appropriate performance index might be of the form

$$J_E = 1/2 \int_0^\infty [(\underline{y} - \underline{y}_m)^T \underline{Y}_E (\underline{y} - \underline{y}_m) + \underline{u}^T \underline{U}_E \underline{u}]dt \qquad \text{(II-7)}$$

In order to achieve a "standard" index [6,30], define

$$\underline{q} = [\underline{x}^T \,\vdots\, \underline{y}_m^T\,]^T \qquad \text{(II-8)}$$

so that

$$\dot{\underline{q}} = \begin{bmatrix} \dot{\underline{x}} \\ \dot{\underline{y}}_m \end{bmatrix} = \begin{bmatrix} \underline{A} & \underline{0} \\ \underline{0} & \underline{A}_m \end{bmatrix} \begin{bmatrix} \underline{x} \\ \underline{y}_m \end{bmatrix} + \begin{bmatrix} \underline{B} \\ \underline{0} \end{bmatrix} \underline{u} \qquad \text{(II-9)}$$

and thus,

$$J_E = 1/2 \int_0^\infty [\underline{q}^T \underline{X}_E \underline{q} + \underline{u}^T \underline{U}_E \underline{u}]dt \qquad \text{(II-10)}$$

where

$$\underline{X}_E = \begin{bmatrix} \underline{C}^T \underline{Y}_E \underline{C} & -\underline{C}^T \underline{Y}_E \\ -\underline{Y}_E \underline{C} & \underline{Y}_E \end{bmatrix} \qquad \text{(II-11)}$$

Since the resulting control law is of the form

$$\underline{u} = - \underline{G}_c \underline{q} = - \underline{G}_{c1} \underline{x} - \underline{G}_{c2} \underline{y}_m \qquad \text{(II-12)}$$

the model output must be simulated in order to generate the required feedforward control.

A very general structural diagram of such an explicit model-follower is shown in Figure II-1. The value of $\underline{G}_{c1}$ is independent of the form of the explicit model to be followed and, for this simple example, is the gain determined by solution of the optimal regulator problem for the controlled system.

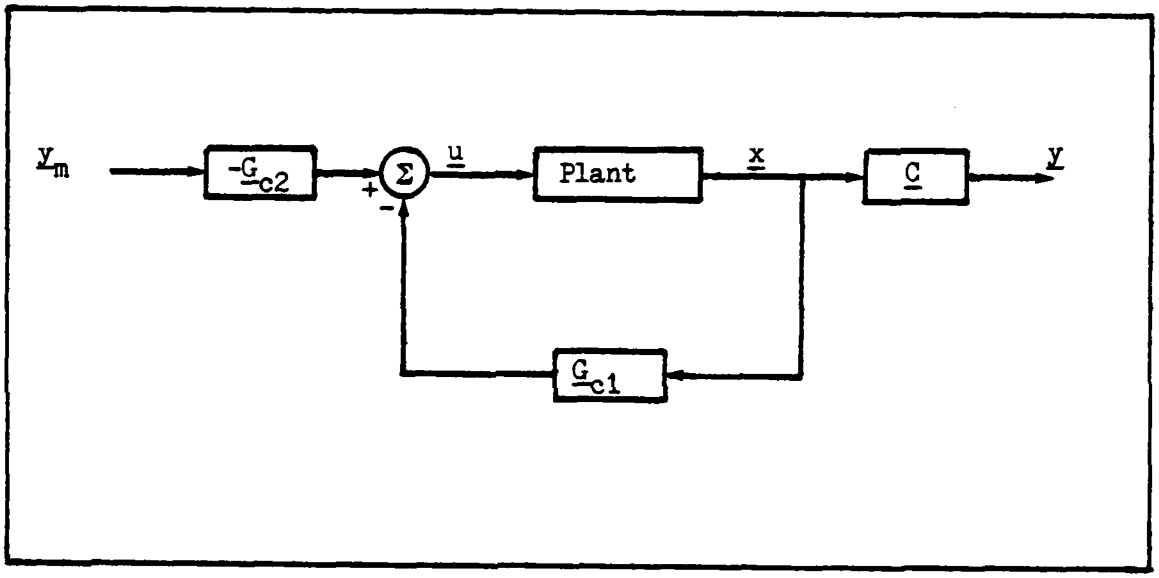


Figure II-1. Explicit Model-Follower, with Simple Feedback Regulator [6]

If the controlled system could be perfectly modelled and operated under ideal conditions without any unmodelled disturbances, then performance specifications could be met with either implicit or explicit model-following. Under more realistic conditions, the two types of controllers have some important differences [16].

The implicit formulation has an effect that is asymptotically (as state weightings grow large) equivalent to eigenstructure assignment

[25], and the matching of the poles of the model and controlled system can produce desirable transient response. Rejection of unmodelled zero-mean disturbances may also be better than with an explicit controller, since good disturbance rejection characteristics can be included in the model response so that the disturbance states need not be modelled in the controller. However, since the feedback gain is directly a function of $(\underline{A} - \underline{A}_m)$, the implicit model-follower is particularly sensitive to parameter variations and plant modelling inaccuracies.

The explicit model-follower is the more complex formulation. Since the model dynamics must be simulated by the controller (which causes a time lag), and because no attempt is made to match the closed-loop poles of the system to the poles of the performance model, a higher feedback gain is generally required than that of the implicit model-follower in order to achieve comparable transient response. Since the actual outputs are compared instead of rates of change, steady-state operation, rejection of constant disturbances, and sensitivity of steady-state output response to parameter variations is generally better with explicit model-following [16,25].

The advantages of both model-following formulations may be achieved by combining the two, resulting in additional flexibility and degrees of design freedom, as well as a more complicated design procedure. The performance index for such a design would be a linear combination of the indices for the implicit and explicit model-followers, such as:

$$J_c = 1/2\int_0^\infty [(\dot{\underline{y}} - \underline{A}_m\underline{y})^T\underline{Y}_I(\dot{\underline{y}} - \underline{A}_m\underline{y}) + (\underline{y} - \underline{y}_m)^T\underline{Y}_E(\underline{y} - \underline{y}_m) + \underline{u}^T\underline{U}_c\underline{u}]dt \qquad \text{(II-13)}$$

A design method based on using the performance index of (II-13) would provide the designer the capability of optimizing the mix of desirable characteristics of the two types of model-followers by adjusting the relative values of $\underline{Y}_I$ and $\underline{Y}_E$, including setting either one to zero.

When command inputs to the system are allowed, both implicit and explicit model-following formulations would require that the command input and outputs be modelled so that appropriate feedforward controls could be generated to drive the system outputs to the appropriate non-zero steady-state values. The implicit model-follower would then also require the use of feedforward control, as was originally the case only with the explicit controller. In combined implicit/explicit model-following, the feedforward controls are dictated jointly by the implicit and explicit performance index terms, but the feedback controls are affected only by the implicit model-following control which attempts to match the system's closed-loop poles with those of the model. Thus the use of implicit model-following is closely tied to the closed-loop system characteristics, including stability robustness. Feedforward control, and thus the use of explicit model-following, will have no effect on the closed-loop stability robustness of a linear system.

2.3 The CGT/PI/KF Controller

The controller of interest in this study consists of a Command Generator Tracker (CGT), which produces the required feedforward controls to attempt to cause the system to track a model trajectory in response to command inputs, and a Proportional-plus-Integral (PI) inner-loop feedback controller to provide regulation and closed-loop stability. With full-state feedback, this configuration will be referred to as CGT/PI; a Kalman Filter (KF) may be added to provide the required state estimates, thus producing the CGT/PI/KF controller. In order to achieve a tractable implementation, both the controller and the filter often employ constant gains, and such was the case in this study. The benefits of the reduced computational burden are assumed to outweigh any degradation that would result from using the constant-gain approximation. A block diagram of the CGT/PI/KF controller is shown in Figure II-2.

This configuration provides a great deal of design flexibility. The CGT is an explicit model-follower; the command model used is the ideal performance model, and the CGT may also contain models of disturbances that the system is to reject. The PI controller may be designed simply to provide optimal "type 1" feedback characteristics, including rejection of unknown disturbances and good transient and steady-state characteristics. The development of this type of "standard" PI regulator is outlined in Appendix A. The PI regulator may also provide implicit model-following characteristics by inclusion of an implicit model in the performance index. The Kalman filter provides state estimates needed by the PI controller as well as estimates of modelled disturbances for use by the CGT in generating optimal feedforward control.

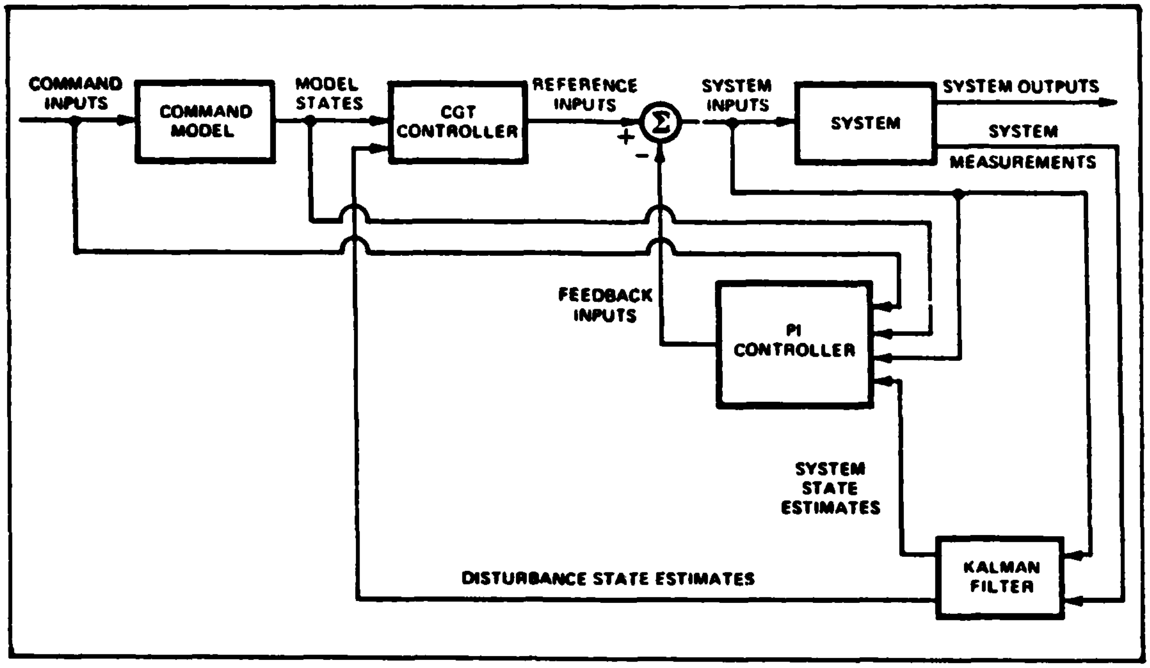


Figure II-2. The CGT/PI/KF Controller [31].

The CGT/PI/KF controller was the subject of recent AFIT theses by R. M. Floyd and A. Moseley [16,34]. The results of those efforts included interactive computer software for the design and performance evaluation of CGT/PI/KF controllers. The referenced works provide complete detail on the development of the design and evaluation procedure as well as guides for using the software, and are highly recommended reading. A review of the basic structure of the controller and the models used in its design using the aforementioned software follows.

The design of the CGT/PI or CGT/PI/KF requires the definition of several different linear models. The designer defines each of these as a continuous-time model. The computer program then converts each of the models to an equivalent discrete-time form [16,29,30] for use in developing the discrete-time digital controller. The design and truth models are mathematical representations of the system to be controlled, and are identical in form. The truth model, at least in theory, embodies all that the designer knows about the system, and is used only to evaluate the performance of the controller. The design model is a purposefully simplified version which describes the system well enough to become the basis for a computationally tractable control law. Hence, the design model is used in the synthesis of the controller, and the truth model is "connected" to the controller for subsequent performance evaluation. The design model used internally by the CGT/PI/KF design software, then, consists of a linear time-invariant discrete-time stochastic vector model of the form

$$\underline{x}(t_{i+1}) = \underline{\Phi}\underline{x}(t_i) + \underline{B}_d\underline{u}(t_i) + \underline{E}_{xd}\underline{n}_d(t_i) + \underline{w}_d(t_i) \tag{II-14}$$

$$\underline{y}(t_i) = \underline{C}\underline{x}(t_i) + \underline{D}_y\underline{u}(t_i) + \underline{E}_y\underline{n}_d(t_i) \tag{II-15}$$

where $\underline{x}(t_i)$ is the system state at the sample time t_i, $\underline{u}(t_i)$ is the control applied to the system at time t_i and held constant until time t_{i+1}, $\underline{n}_d(t_i)$ represents the time-correlated disturbances affecting the system, $\underline{w}_d(t_i)$ is a zero-mean, discrete-time white Gaussian driving noise of covariance $\underline{Q}_d$, and $\underline{y}(t_i)$ is the output vector over which control is to be exercised. Such a model could, in general, represent an actual discrete process, but in this case is an equivalent discrete-time description of an underlying continuous-time process [16,30], assuming the nonsingularity of $\underline{\Phi}$. If the actual system is nonlinear, the model may represent linearized perturbation states, as in the case of aircraft equations expanded about a trim condition.

The time-correlated disturbance vector may be generated as the output of a time-invariant linear shaping filter:

$$\underline{n}_d(t_{i+1}) = \underline{\Phi}_n \underline{n}_d(t_i) + \underline{w}_{nd}(t_i) \tag{II-16}$$

with $\underline{w}_{nd}(t_i)$ a zero-mean white Gaussian noise of covariance $\underline{Q}_{nd}$ that is independent of $\underline{w}_d(t_i)$ in (II-14).

The model for the system's desired dynamic response, used for explicit model-following, is called the command generator model:

$$\underline{x}_m(t_{i+1}) = \underline{\Phi}_m \underline{x}_m(t_i) + \underline{B}_{md} \underline{u}_m(t_i) \tag{II-17}$$

$$\underline{y}_m(t_i) = \underline{C}_m \underline{x}_m(t_i) + \underline{D}_m \underline{u}_m(t_i) \tag{II-18}$$

With Floyd and Moseley's software, the command generator model is also used as the performance model for implicit model-following; however, it is possible to use different model definitions for the two design functions. In fact, such a generalization was employed in this research. The model control, $\underline{u}_m(t_i)$, is the actual command input to the controller, as from the pilot stick, and is assumed to vary slowly with time compared to the controller sampling period; with the sampling rates used in modern digital controllers, $\underline{u}_m(t_i)$ may be considered to be piecewise constant for the purpose of control law derivation [30].

In the design implementation developed by Floyd and Moseley [16,34], the use of explicit model-following requires that the dimension of $\underline{y}_m$ be the same as that of $\underline{y}$; the use of implicit model-following requires that the dimension of $\underline{y}$ be the same as that of

$\underline{x}_m$. The only other dimensionality restriction is that the number of control inputs equal the number of controlled outputs. Designs may be conducted which actually violate the last restriction by simply introducing a "dummy" input or output. A true solution is possible only when the number of controls equals or exceeds the number of inputs; an approximation based on a matrix pseudoinverse results otherwise [16,30].

Optimal controller designs based on the LQG assumptions are said to possess the property of certainty equivalence [30]. Because of this property, the optimal stochastic controller for such a design consists of an optimal linear Kalman filter cascaded with a deterministic optimal linear controller, and the design of the filter and of the controller can be conducted separately [30]. Certainty equivalence is invoked at this point in the CGT/PI/KF development; the design of the deterministic CGT/PI follows.

In the deterministic open-loop formulation, the CGT must force the error between the model output and the system output to be zero:

$$\underline{e}(t_i) = \underline{y}(t_i) - \underline{y}_m(t_i) = \underline{0} \qquad \text{(II-19)}$$

Ideal state and control trajectories can be defined as the time histories of system states and controls that must be followed to accomplish this objective. First, the ideal trajectory must obey the basic state model relationships of (II-14), or, in a deterministic setting,

$$\underline{x}_I(t_{i+1}) = \underline{\Phi}\, \underline{x}_I(t_i) + \underline{B}_d \underline{u}_I(t_i) + \underline{E}_{xd} \underline{n}_d(t_i) \qquad \text{(II-20)}$$

If the ideal trajectories are constrained to be linear combinations of

$\underline{x}_m(t_i)$, $\underline{u}_m(t_i)$, and $\underline{n}_d(t_i)$, i.e.,

$$\begin{bmatrix} \underline{x}_I(t_i) \\ \underline{u}_I(t_i) \end{bmatrix} = \begin{bmatrix} \underline{A}_{11} & \underline{A}_{12} & \underline{A}_{13} \\ \underline{A}_{21} & \underline{A}_{22} & \underline{A}_{23} \end{bmatrix} \begin{bmatrix} \underline{x}_m(t_i) \\ \underline{u}_m(t_i) \\ \underline{n}_d(t_i) \end{bmatrix} \qquad \text{(II-21)}$$

then the CGT solution is achieved by solving for the constant matrix partitions in (II-21) such that (II-19) and (II-20) are also satisfied. Defining a $\underline{\Pi}$ matrix as

$$\begin{bmatrix} \underline{\Pi}_{11} & \underline{\Pi}_{12} \\ \underline{\Pi}_{21} & \underline{\Pi}_{22} \end{bmatrix} = \begin{bmatrix} \underline{\Phi} - \underline{I} & \underline{B}_d \\ \underline{C} & \underline{D}_y \end{bmatrix}^{-1} \qquad \text{(II-22)}$$

it can be shown [30] that the solution is

$$\underline{A}_{11} = \underline{\Pi}_{11}\underline{A}_{11}(\underline{\Phi}_m - \underline{I}) + \underline{\Pi}_{12}\underline{C}_m \qquad \text{(II-23a)}$$

$$\underline{A}_{12} = \underline{\Pi}_{11}\underline{A}_{11}\underline{B}_{md} + \underline{\Pi}_{12}\underline{D}_m \qquad \text{(II-23b)}$$

$$\underline{A}_{13} = \underline{\Pi}_{11}\underline{A}_{13}(\underline{\Phi}_n - \underline{I}) - \underline{\Pi}_{11}\underline{E}_{xd} - \underline{\Pi}_{12}\underline{E}_y \qquad \text{(II-23c)}$$

$$\underline{A}_{21} = \underline{\Pi}_{21}\underline{A}_{11}(\underline{\Phi}_m - \underline{I}) + \underline{\Pi}_{22}\underline{C}_m \qquad \text{(II-23d)}$$

$$\underline{A}_{22} = \underline{\Pi}_{21}\underline{A}_{11}\underline{B}_{md} + \underline{\Pi}_{22}\underline{D}_m \qquad \text{(II-23e)}$$

$$\underline{A}_{23} = \underline{\Pi}_{21}\underline{A}_{13}(\underline{\Phi}_n - \underline{I}) - \underline{\Pi}_{21}\underline{E}_{xd} - \underline{\Pi}_{22}\underline{E}_y \qquad \text{(II-23f)}$$

and methods have been developed [3,23] to solve these efficiently. The ideal constant-gain feedforward control is therefore a linear combination of the command generator model states, the command generator model (command) input vector, and the disturbance states [16,30,34]. Once (II-23) is solved, the lower partition of (II-21) yields $\underline{u}_I(t_i)$ as

$$\underline{u}_I(t_i) = \underline{A}_{21}\underline{x}_m(t_i) + \underline{A}_{22}\underline{u}_m(t_i) + \underline{A}_{23}\underline{n}_d(t_i) \tag{II-24}$$

When a deterministic, constant-gain PI controller, as developed in Appendix A, is included to provide feedback control, the control law becomes [16,30]

$$\begin{aligned}\underline{u}(t_i) = \underline{u}(t_{i-1}) &- \underline{K}_x[\underline{x}(t_i) - \underline{x}(t_{i-1})] \\ &+ \underline{K}_z\left\{[\underline{C}_m \quad \underline{D}_m]\begin{bmatrix}\underline{x}_m(t_{i-1}) \\ \underline{u}_m(t_i)\end{bmatrix} - [\underline{C} \quad \underline{D}_y]\begin{bmatrix}\underline{x}(t_{i-1}) \\ \underline{u}(t_{i-1})\end{bmatrix}\right\} \\ &+ \underline{K}_{xm}[\underline{x}_m(t_i) - \underline{x}_m(t_{i-1})] \\ &+ \underline{K}_{xu}[\underline{u}_m(t_i) - \underline{u}_m(t_{i-1})] \\ &+ \underline{K}_{xn}[\underline{n}_d(t_i) - \underline{n}_d(t_{i-1})]\end{aligned} \tag{II-25}$$

where, in terms of the PI regulator development in Appendix A and (II-22) and (II-23),

$$\underline{K}_x = \underline{G}_{c1}\underline{\Pi}_{11} + \underline{G}_{c2}\underline{\Pi}_{21} \tag{II-26a}$$

$$\underline{K}_z = \underline{G}_{c1}\underline{\Pi}_{12} + \underline{G}_{c2}\underline{\Pi}_{22} \tag{II-26b}$$

$$\underline{K}_{xm} = \underline{K}_x\underline{A}_{11} + \underline{A}_{21} \tag{II-26c}$$

$$\underline{K}_{xu} = \underline{K}_x\underline{A}_{12} + \underline{A}_{22} \tag{II-26d}$$

$$\underline{K}_{xn} = \underline{K}_x\underline{A}_{13} + \underline{A}_{23} \tag{II-26e}$$

The time argument of the $\underline{u}_m(t_i)$ term in the second line of (II-25) is a consequence of the need to ensure consistency in the equations for the ideal state trajectory (II-20 and II-21) as the system undergoes a change in the value of the model input, $\underline{u}_m$. Such a change requires that the time argument of the model input be advanced by one sample period in (II-17), providing direct feedthrough of $\underline{u}_m$ to $\underline{x}_m$. While this precludes precomputation of $\underline{x}_m(t_i)$ in the background between sampling points, it also improves the initial transient performance of the system by speeding up the command model by one sampling period [16,30].

The final step in the development of the overall CGT/PI/KF is the incorporation of a standard steady-state, constant-gain Kalman filter to provide estimates of the system states and disturbances. Due to the certainty equivalence property of LQG designs discussed earlier, this does not change the form of the control law; $\underline{x}(t_i)$ and $\underline{n}_d(t_i)$ are simply replaced by the filter estimates [30]. The dynamics model for the filter consists of the design model for system dynamics augmented with the time-correlated disturbance model, and the measurements are modelled as

$$\underline{z}(t_i) = [\underline{H} \quad \underline{H}_n] \begin{bmatrix} \underline{x}(t_i) \\ \underline{n}_d(t_i) \end{bmatrix} + \underline{v}(t_i) \tag{II-27}$$

where $\underline{v}(t_i)$ is a zero-mean, discrete-time white Gaussian noise of covariance $\underline{R}$.

The calculation of the various gains in the final control law requires extensive matrix algebra and a degree of trial-and-error iteration with varying performance index weights, all of which is the purpose of the software developed by Floyd and Moseley. To use the software, the designer must specify the models and performance index weights; these include weights on output deviations, inputs and input rates, plus weights on output rate deviations when implicit model-following is desired. The program CGTPIF [16,34] will solve for all of the required gains and conduct a performance analysis of each of the individual components of the controller. If a Kalman filter is designed, the program PFEVAL [34] can be run to conduct a covariance analysis of the entire controller/filter structure as a unit, as well as a statistical analysis of a controller that assumes perfect access to all states (to display the impact of the filter on performance explicitly). Both programs allow performance analysis using a linear truth model to evaluate the controller in a realistic environment. To evaluate controller robustness, a truth model which differs substantially (in dimensionality as well as coefficient values) from the design model may be used. The controller design software (CGTPIF) written by Floyd [16] and modified by Moseley [34] was used with only minor modifications in this study; the version used is documented in Appendix D.

2.4 Summary

In this chapter, the use of model-following was shown to provide a means of incorporating conventional time-domain specifications into controller designs achieved through the use of modern, LQ design methods. The structure of a particularly useful model-following configuration, the CGT/PI/KF controller, was outlined in some detail. The employment of implicit model-following in controller design directly affects the closed-loop characteristics of the resulting system, including stability robustness. In the chapter that follows, methods of analyzing and enhancing the stability robustness of MIMO control systems are presented.

III. Robustness Analysis and Enhancement

3.1 Introduction

This chapter deals with methods for analyzing the stability robustness in MIMO control systems, as well as techniques that can help improve the robustness characteristics of LQG controllers. Singular value analysis is shown to be an intuitively pleasing multi-loop generalization of classical SISO design techniques that can be applied to both stability robustness and performance characteristics. The use of implicit model-following to affect the eigenstructure of the closed-loop control system is discussed as a means of robustness enhancement. Finally, a method for recovering a degree of the robustness lost in replacing full-state feedback with Kalman filter state estimates is introduced.

3.2 Singular Value Analysis

Consider the simple continuous-time, linear, time-invariant control system shown in Figure III-1. The plant is represented by the $\underline{G}$ transfer function, $\underline{K}$ is the controller, $\underline{r}$ is the forcing command, $\underline{y}$ is the output, and $\underline{n}$ and $\underline{v}$ are, respectively, disturbances and measurement noise. Although the disturbances are depicted as entering the system at the plant output, it is obvious that disturbances might as well be shown entering at the control input; the choice here is merely for simplification of the presentation. Also, this structure assumes that noise corrupted measurements of all outputs are available. While a unity feedback configuration is depicted, the addition of a precompensator ($\underline{P}$) allows generalization to an equivalent unity feedback representation of a non-unity feedback controller [14], including such configurations as the CGT/PI/KF.

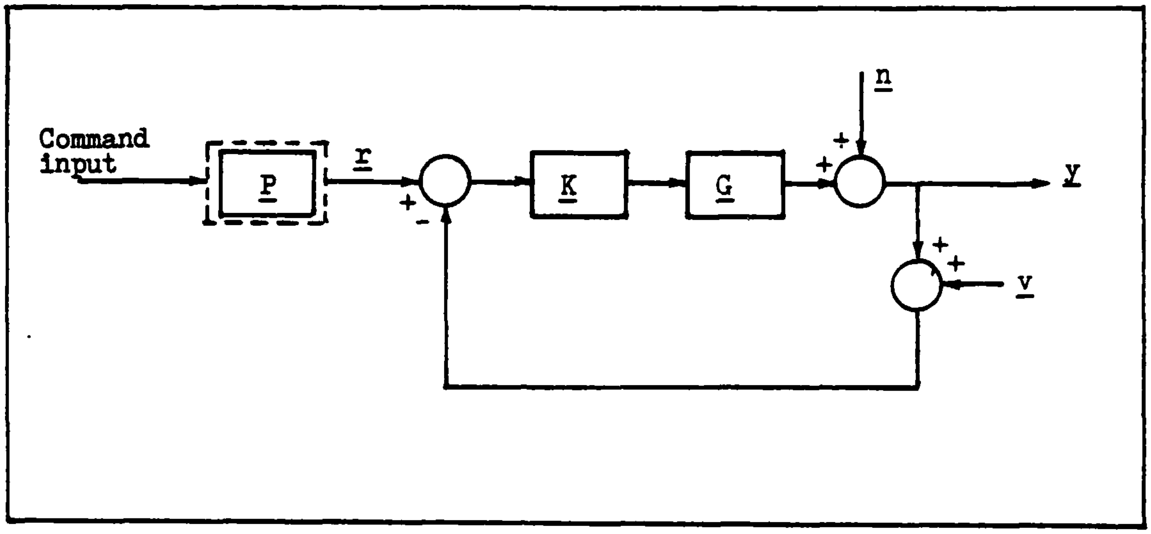


Figure III-1. Typical Control System [14].

As stated in Chapter II, the precompensator is outside of the loop, so it will not affect the closed-loop characteristics, including robustness, of this linear system. The output of this control system is

$$\underline{y} = \underline{GK}(\underline{I} + \underline{GK})^{-1}(\underline{r}-\underline{v}) + (\underline{I} + \underline{GK})^{-1}\underline{n} \tag{III-1}$$

Now consider the familiar SISO case in which all of the above quantities are scalar. The complex scalar gk is referred to as the loop gain of the system, and the complex function (1 + gk) is the system's return difference function. Classical design theory states that the system's command tracking, disturbance rejection and sensitivity to modelling errors will be good over the range of input frequencies where (the determinant) $|1 + gk|$ is large, or where $|gk|$

is much larger than 1 (i.e., the system bandwidth) [43]. Designing for this consideration is limited by the response to measurement noises (the same as to commands) and stability [43]. The stability restriction on the loop gain is commonly stated in terms of the Nyquist Criterion: the number of counterclockwise encirclements of the (-1 + j0) point in the complex plane realized by the function $gk(j\omega)$ must be equal to the number of unstable modes (right-half-plane zeros) of the function [8]. This is the same as the requirement that the system possess positive gain and phase margins; since most physical systems can be described by transfer functions with a pole-zero excess of at least two [21], the frequency range over which the magnitude of the loop gain may exceed unity is limited by the frequency at which the phase angle exceeds 180 degrees.

At this point it is necessary to consider the existence of modelling errors and parameter uncertainty in the plant representation (g). If the actual value of g differs from the design model value by an amount up to and including Δg, then the possibility exists that the actual system's performance will be degraded. An even more serious possibility is the loss of stability due to the perturbation. Performance degradation can be assessed by reevaluating the bandwidth of the loop gain using the true (in the worst case) plant, $g' = g + \Delta g$; stability can only be guaranteed if no value of g' in the range specified causes the number of Nyquist encirclements to change.

In many cases, a designer will have some insight into the types and magnitudes of parameter uncertainties in the design model; they may be the result of intentionally ignored modes, nonlinearities or time variations. As might be expected, special design and analysis methods can be applied when the structure of the modelling uncertainty is known [12,15,18,27]. A most general class of uncertainties (norm-bounded but otherwise unconstrained, or "unstructured") will be considered in this thesis. In most cases, the low frequency characteristics of physical systems are relatively well known and are readily expressed using linear or linearized perturbation models. At higher frequencies, system characteristics generally become more nonlinear and less predictable. The perturbations to the design model frequently exceed the magnitude of the nominal transfer function at high frequencies, and phase uncertainties may exceed ±180 degrees [14]. The need to ensure stability in the face of such uncertainties is reflected in the phase and gain margin requirements of classical SISO design. Even in the SISO case, however, analysis of gain and phase margins does not fully reflect the true stability robustness of a control system, because it does not indicate the minimum amount of simultaneous gain and phase variation required to produce instability. The Nichols plot [8] of Figure III-2 is an example of a system with good gain and phase margin, but very poor robustness, since a very small simultaneous change in gain and phase could produce instability.

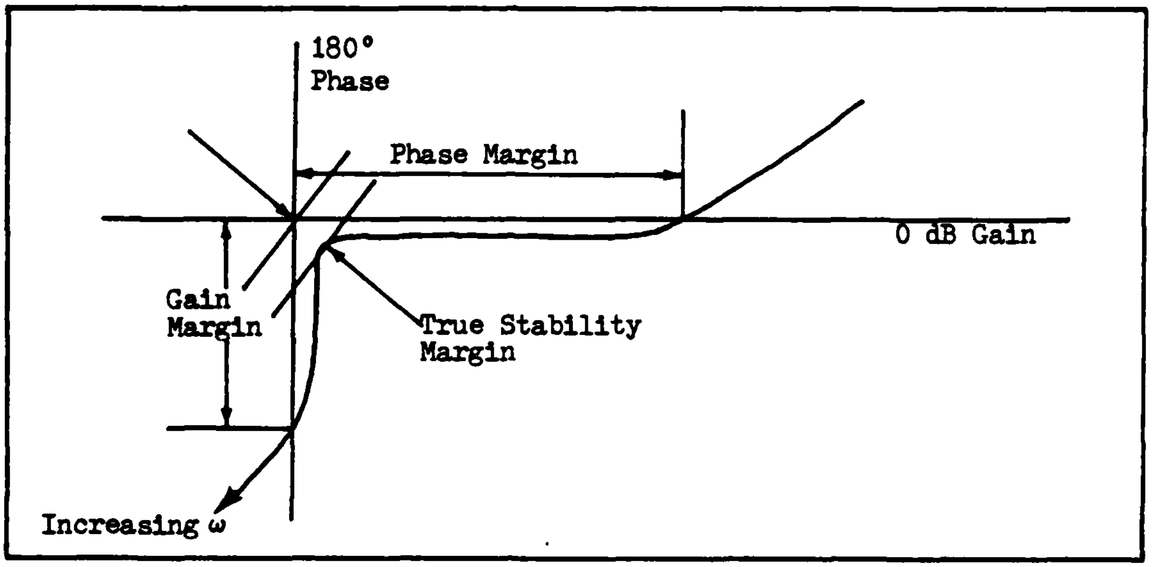


Figure III-2. Nichols Plot Displaying Good Gain and Phase Margins, but Poor Robustness.

In the case of multi-loop control systems, an analysis of stability robustness that parallels that of the SISO case is complicated further by the need to consider simultaneous phase and gain variations in all possible combinations of loops. This is due to the fact that, in most cases, the control loops are coupled, and changes in one loop will affect other loops in the system. Since such an analysis would be a monumental, if not impossible, task for systems consisting of many loops, a more powerful tool is needed. In particular, a means of assessing the "size" of the $\underline{G}$ and $\underline{K}$ matrices and their mapping effect on the system inputs and outputs (analogous to scalar gains) will be useful in generalizing the scalar concepts already discussed.

The spectral norm $\|\bullet\|$ for an n-by-n matrix $\underline{A}$ is defined by

$$\|\underline{A}\| = \max_{\|\underline{x}\|=1} \|\underline{A}\,\underline{x}\| = \max_{\|\underline{x}\|\neq 1} \|\underline{A}\,\underline{x}\| / \|\underline{x}\| \qquad \text{(III-2)}$$

and is a measure of the "size" of the matrix and its mapping effect on the vector $\underline{x}$ [30]. The singular values, σ_1 through σ_n, of the (possibly) complex matrix $\underline{A}$ are defined as the non-negative square roots of the eigenvalues of $\underline{A}^*\underline{A}$, where $\underline{A}^*$ is the conjugate transpose of $\underline{A}$ [11]. The singular values of $\underline{A}$ and the norm $\|\underline{A}\|$ are related by the inequalities

$$\sigma_{\min}(\underline{A}) \overset{\Delta}{=} \underline{\sigma}(\underline{A}) = 1/\|\underline{A}^{-1}\| \tag{III-3}$$

$$\sigma_{\max}(\underline{A}) \overset{\Delta}{=} \bar{\sigma}(\underline{A}) = \|\underline{A}\| \tag{III-4}$$

so that knowledge of the minimum and maximum singular values of a matrix provides information as to the maximum "gain" effect it can have as well as its closeness to singularity [30]. This information cannot be obtained through analysis of the eigenvalues of $\underline{A}$; although, for all eigenvalues of $\underline{A}$,

$$\underline{\sigma} \leq |\lambda| \leq \bar{\sigma} \tag{III-5}$$

the smallest eigenvalue may be much larger than $\underline{\sigma}$ [11]. It is possible to reduce any n-by-n nonsingular matrix to the form

$$\underline{A} = \underline{U}\,\underline{\Sigma}\,\underline{V} \tag{III-6}$$

where $\underline{U}$ and $\underline{V}$ are unitary matrices and $\underline{\Sigma}$ is a diagonal matrix whose elements are the singular values of $\underline{A}$. The process is called singular value decomposition, and can be performed by available computer software [9,24].

Now consider the MIMO case for the control system of Figure III-1. The scalar loop gain of the SISO system is replaced by a matrix function, and now a distinction must be made as to where the loop is broken. If the loop is cut at a point between the controller and the plant, then the loop gain is $\underline{KG}$; if it is cut at the plant output, the loop gain is $\underline{GK}$. In general, the two will be different. The return difference scalar function is also replaced by a return difference matrix, which also depends on the point at which the loop is cut. The role of these matrices in performance and stability analysis is analogous to that of their scalar counterparts, but due to the multivariable nature of the problem, such analysis is more complicated. In the MIMO case, the mapping effect of the loop and return difference matrices is dependent not only on the matrices themselves, but on the direction of the vector quantity that is being mapped. As a related example, the eigenvalues of a multi-loop system define the modes of the system's response to any excitation, while its eigenvectors determine the distribution of the response energy based on the relative directional orientation of the excitation and the responding loops. Singular value analysis provides insight into the minimum and maximum effects of a matrix function, so that the most pessimistic prospects for performance and robustness can be assessed. While pessimism is not necessarily a bad thing, the conservatism inherent in this measure can in some circumstances be arbitrarily large, thus rendering it somewhat useless. The degree to which this is true is a function of the problem being analyzed [12].

The minimum singular value of the system's loop gain matrix can be used to assess the minimum bandpass of any loop in the system for inputs which occur in an arbitrary direction. The magnitude of the minimum singular value is a measure of system performance; good performance generally results when this value is large over the desired frequency range [14]. The maximum singular value is an estimate of the maximum response in any loop to an arbitrary input and, as with single-loop designs, it is important to ensure that this value is small (much less than 1) at frequencies where significant uncertainty exists. In effect, the low frequency performance specifications and high frequency stability requirements provide boundaries within which the designer attempts to restrain the loop gain matrix singular values, as shown in Figure III-3 [14]. While this concept provides some ability to evaluate limits of performance and absolute stability, a more concise measure of the amount of uncertainty which can be tolerated in the loop without causing instability is required as a true measure of stability robustness.

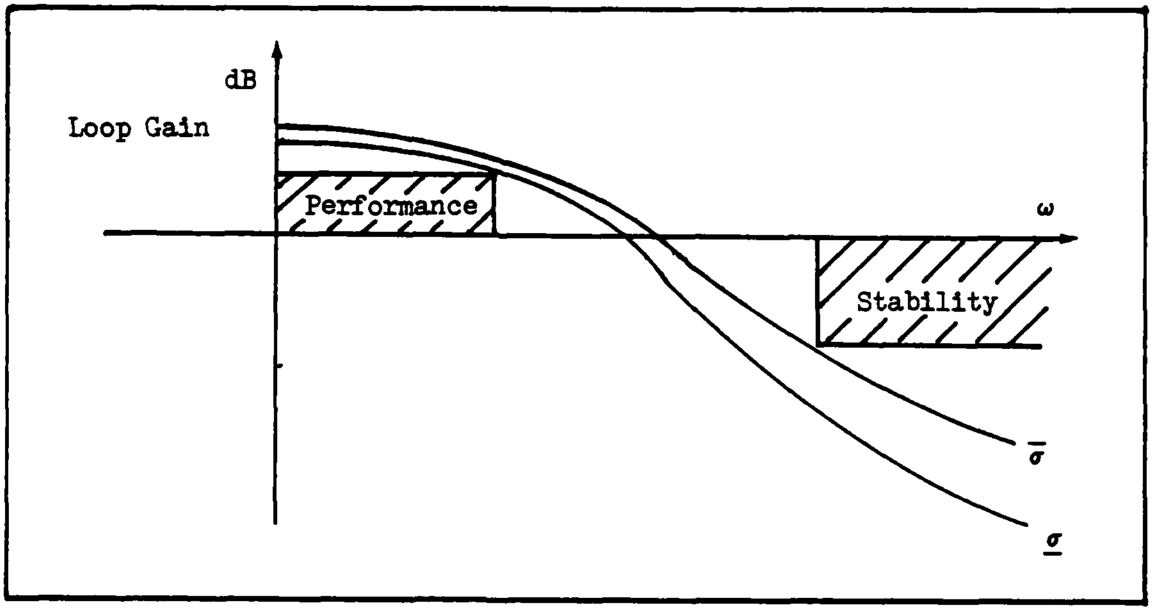


Figure III-3. Design Boundaries for Singular Values [14].

If the nominal system is stable and the uncertainty can be represented as a multiplicative alteration (the type which corresponds to the classical concepts of gain and phase margins) of the form $(\underline{I} + \underline{L})$, so that the perturbed loop gain (with the loop cut at the control input) is

$$\underline{KG}' = (\underline{I} + \underline{L})\ \underline{KG} \qquad \text{(III-7)}$$

then it can be shown [11,14,30] that no perturbation $\underline{L}$ such that $\bar{\sigma}\ (\underline{L}) < \ell$ will change the number of Nyquist encirclements of the loop gain as it is allowed to vary from $\underline{KG}$ to $\underline{KG}'$ if it is true that

$$\ell \leq 1/\ \bar{\sigma}(\underline{I} - [\underline{I} + \underline{KG}]^{-1}) \qquad \text{(III-8)}$$

Employing the matrix inversion lemma [30] which states that, for invertible $\underline{A}$,

$$(\underline{I} + \underline{A}^{-1})^{-1} = \underline{I} - (\underline{I} + \underline{A})^{-1} \qquad \text{(III-9)}$$

as well as (III-3) and (III-4), (III-8) is equivalent to

$$\ell \leq \underline{\sigma}(\underline{I} + [\underline{KG}]^{-1}) \qquad \text{(III-10)}$$

or

$$\bar{\sigma}(\underline{L}) < \underline{\sigma}(\underline{I} + [\underline{KG}]^{-1}) \qquad \text{(III-11)}$$

for all frequencies (in either the z- or s-domain [8,30]). The quantity $(\underline{I} + [\underline{KG}]^{-1})$ is called the inverse return difference matrix function for the system. The maximum value of ℓ for which the inequality in (III-10) holds is a measure of the stability robustness of the closed-loop control system. Use of this measure will be made in the remainder of this thesis; expressions for the loop gain and inverse return difference matrices for the CGT/PI/KF are developed in Appendix B.

Analysis of the effects of such arbitrary perturbations, or "unstructured singular value analysis," has a major drawback in that it considers perturbations that might never physically occur; the robustness estimates are therefore conservative. If a designer has considerable information about the possible structure of uncertainties likely to occur, then in many cases much tighter and more realistic bounds can be place on robustness estimates through the use of more complex methods of "structured singular value analysis" [12,15,18,27]. These will not be pursued further in this thesis effort, but are potentially fruitful for future research.

3.3 Analysis Through Simulation

An excellent alternative method of assessing the robustness of a control system design is through actual simulation, in which the control law is allowed to operate on a truth model. The truth model can be altered to represent any number of physically feasible perturbations to the design model, including previously ignored dynamics and nonlinearities. With the use of simulation, there is no issue of conservatism, and the stability of the control system with respect to the perturbed model can be fully evaluated for any and all physically important perturbations. However, there is an infinite number of possible perturbed models to be considered, so considerable judgment is required in the conduct of such analysis. This more conventional means of robustness evaluation was used extensively in this study, both as a complement to singular value analysis and as a means of assessing the usefulness of singular value analysis.

3.4 Robustness Enhancement Methods

The primary design tool for robustness enhancement that was to be employed in this study was the use of implicit model-following to affect the eigenstructure of the closed-loop control system. Gilbert demonstrated that the sensitivity of the characteristic roots of a matrix to variations in the matrix elements is minimized when the nominal eigenvectors of the matrix are made as nearly orthogonal as possible [19]. Since the designer has a considerable amount of freedom in assigning the eigenstructure of a multivariable system [1,33], choosing closed-loop system eigenvectors that are maximally orthogonal will, in general, produce robustness benefits in that the system will be less sensitive to parameter variations. Many techniques are available that can be used in control system eigenstructure assignment. Broussard and Berry have shown that one such method is the use of implicit model-following [6]. The minimization of an LQ performance index that is based on an implicit model, as shown in Section 2.2, has a direct effect on the closed-loop system characteristics. This is because the feedback control causes the eigenvalues and eigenvectors of the closed-loop system to tend to match those of the implicit model. With the CGT/PI/KF controller configuration as formulated by Floyd and Moseley [16,34], the designer can choose not only the implicit model to be used (which need not be the same as the CGT command model), but the relative weighting in the performance index given to implicit versus explicit model-following control. By choosing an implicit model for the regulator design with desired eigenvalues and orthogonal eigenvectors, increasing the relative weighting on implicit model-following, and by matching a greater number of system states to a desirable implicit model, the designer should, to some extent, be able to approach an optimally robust eigenstructure in the controlled system. This method of eigenstructure assignment would be one which provides a great deal of insight for the designer in progressing through an iterative design process. It should be noted that various other methods of achieving eigenstructure assignment are discussed in current literature [1,20].

Full-state LQ optimal steady-state feedback laws have been shown to possess impressive guaranteed robustness characteristics. For continuous-time controllers (and in the limit, as sampling time goes to zero, for sampled-data controllers) these guarantees include infinite simultaneous gain margins and at least 60 degree phase margins in all loops [30,39]. Since they are valid only for linear systems with full-state feedback, these guarantees are largely theoretical. Real systems are not, in general, linear; nor are they finite-dimensional. Thus, the concept of linear, full-state feedback is itself somewhat fictitious. Even when large gain and phase margins can be achieved, stability under simultaneous gain and phase variations is not guaranteed, as shown in Section 3.2. Despite the preceding criticism of theoretical robustness guarantees, when the linear model of the system is adequate, full-state LQ feedback controllers, based on such a model, are inherently quite robust [30].

Even when an extremely robust full-state feedback design is achievable, the introduction of an observer such as the Kalman filter into the system to provide estimates of states that are not perfectly accessible produces a degradation in stability robustness. There are no robustness guarantees, even theoretical ones, for LQG controllers [10]. This is due to the dependence of the filter on an internal system model [12,30]. Doyle and Stein [13] have shown that the lost robustness may be asymptotically recovered in continuous-time, minimum phase control systems (at the expense of optimal filter performance under design conditions) by the systematic introduction of a pseudo-noise into the model upon which the filter is based, at the entry points of the controls. An extension of this technique for discrete systems was produced by Capt Lloyd [28]. To apply this technique, the $\underline{Q}_{da}$ term, representing the strength of the noises driving an augmented system model based on (II-14) and (II-16), or

$$\underline{Q}_{da} = \begin{bmatrix} \underline{Q}_d & \underline{0} \\ \underline{0} & \underline{Q}_{nd} \end{bmatrix} \tag{III-12}$$

which occurs in the covariance propagation equation for the Kalman filter based on the same model [29]:

$$\underline{P}(t_{i+1}^-) = \underline{\Phi}_a \underline{P}(t_i^+) \underline{\Phi}_a^T + \underline{Q}_{da} \tag{III-13}$$

where

$$\underline{\Phi}_a = \begin{bmatrix} \underline{\Phi} & \underline{0} \\ \underline{0} & \underline{\Phi}_n \end{bmatrix} \tag{III-14}$$

is replaced by a quantity such as [28]

$$\underline{Q}_{da} + q^2 \underline{BVB^T} \Delta t \tag{III-15}$$

where Δt is the controller sampling time interval, $\underline{B}$ is the continuous-time control matrix, and $\underline{V}$ is a positive definite matrix which the designer may choose to affect the relative rates of recovery in various loops. In effect, this method causes some of the filter poles to migrate, as the value of q is increased, toward stable plant zeroes. The robustness lost due to the presence of the filter can thus be asymptotically recovered, as q is allowed to approach infinity, in a number of control system loops equal to the number of filter measurements [43]. A dual to this approach was given by Stein and Sandell [43] who credit it to Kwakernaak [26]; this method involves systematically increasing the quadratic state weightings in the regulator design in a manner analogous to (III-15), causing the regulator poles to migrate. While Kalman filter designs and robustness recovery techniques were not pursued in this study, a concurrent effort by Lt Jean Howey [22] has centered on such procedures, and should be considered as complementary required reading.

3.5 Summary

In this chapter, methods of analyzing and enhancing the stability robustness of MIMO control systems was presented. Employment of such methods constituted a major portion of the work in this study. The next chapter outlines in more detail how this work was carried out.

IV. Design Objectives, Models and Observations

4.1 Introduction

This chapter outlines more specifically the design objectives undertaken in this study, the models used, and the analysis methods that were employed in evaluating the tentative designs. Observations that emerged during the course of the design iterations involving implicit model-following regulators, as opposed to the "standard" PI regulator formulation, are also included. It is hoped that these observations will provide some insight into the design paths subsequently chosen in this study, and that they will help the reader to understand the design results detailed in Chapter V. It was originally hoped that these observations would also be instructive for designers who might choose to attempt implicit model-following regulator designs. However, the extent to which some of the ideas are true has been affected by the error discovered in the design software. The reader is referred to Appendix E for more information regarding designs using the corrected software.

4.2 Design Models and Objectives

The overall objective of this study was to try to use implicit model-following in the design of a multivariable control system, as a means of enhancing controller capability and robustness in the face of a variety of realistic uncertainties. The primary design tool which was to be used was the software (CGTPIF) written by Floyd and Moseley for the interactive design and analysis of CGT/PI/KF controllers [16,34]. Minor corrections and modifications made to that software for the purposes of this study are documented in Appendix D; the version used herein will henceforth be referred to as CGTPIV (CGT/PI design program, Variant).

A CGT/PI controller for the Advanced Fighter Technology Integration (AFTI) F-16 was chosen as the example controller to be designed. The AFTI F-16 is an aircraft which has been modified for advanced flight control research. Through the use of two independently controlled longitudinal flight control surfaces (a horizontal tail and a trailing edge wing flap), it is possible to achieve direct control of the aircraft's pitch attitude without changing its flight path angle and, hence, its flight trajectory. This capability is expected to be very useful in situations such as air-to-air gunnery, in which the attacker must match the target's trajectory while simultaneously achieving the required gun lead angle in the pitch plane. The design of a pitch-pointing controller for the AFTI F-16 was the primary design example used in previous theses involving the CGT/PI/KF controller configuration and design/evaluation software [16,34]. Such a design was of interest for this study, since it inherently requires the use of MIMO design methodology to achieve decoupled control of the two outputs and maintain closed-loop stability with a plant (the F-16 aircraft) which is unstable.

The efforts of this study used the design model for the AFTI F-16 which was developed in Capt Floyd's thesis [16]. The model represents the linearized flight characteristics of the aircraft operating at .8 mach at 10,000 feet. It is a time-invariant, five-state model in the form

$$\underline{\dot{x}} = \underline{Ax} + \underline{Bu} \qquad \text{(IV-1)}$$

where the state vector $\underline{x}$ consists of

$x(1)$ = pitch angle (degrees)

$x(2)$ = angle of attack (degrees)

$x(3)$ = pitch rate (degrees per second)

$x(4)$ = horizontal tail deflection (degrees)

$x(5)$ = trailing edge flap deflection (degrees)

$x(1) - x(2)$ = flight path angle (degrees) (IV-2)

and

$$\underline{A} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -1.08\text{E-}3 & -1.7 & 0.994 & -0.179 & -0.295 \\ 0 & 5.93 & -0.668 & -25.3 & -5.88 \\ 0 & 0 & 0 & -20.0 & 0 \\ 0 & 0 & 0 & 0 & -20.0 \end{bmatrix} \qquad \text{(IV-3)}$$

$$\underline{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 20.0 & 0 \\ 0 & 20.0 \end{bmatrix} \qquad \text{(IV-4)}$$

Note that, in this design model, actuators are represented as having characteristics of a first-order lag response to commands. The outputs are expressed as

$$\underline{y} = \underline{C}\underline{x} \tag{IV-5}$$

where

$$\underline{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \end{bmatrix} \tag{IV-6}$$

In other words, the outputs of interest are the pitch angle and the flight path angle.

This model was used as the basis for both controller design and initial design evaluation using the CGTPIV software. For subsequent further evaluations, alternate truth models were developed which differed from the design model in terms of flight characteristics and in the inclusion of higher-order and nonlinear actuator dynamics. Various command models were used throughout this study, depending on the application and objective. Since the command model was a designer-selected parameter, each of the command models used will be discussed as such in Chapter V, along with the designs in which they were employed.

Specifically, the design objective was to design a CGT/PI pitch-pointing controller wherein the PI regulator was to be designed using the implicit model-following concept. The feasibility of an "all-implicit" regulator was to be determined, and the performance and robustness characteristics of such a design investigated. In particular, the utility of implicit model-following in enhancing controller robustness in the face of various forms of uncertainty was to be evaluated. The baseline for evaluating the relative "goodness" of the implicit designs was the CGT/PI controller, developed in Capt Floyd's thesis [16], which was based on the design model just defined. That controller design, which did not employ implicit model-following, was duplicated and extensively evaluated early in this study. Its design parameters, performance characteristics and limitations are discussed at length in Chapter V.

4.3 Analysis Methods

The controllers developed in this study were evaluated by various methods to determine their suitability, both in an absolute sense and in comparison to the example standard CGT/PI design used as the baseline.

A computer program called CGTSVD (CGT Singular Value Decomposition) was developed which calculates the minimum and maximum singular values of the loop gain and inverse return difference matrices of a CGT/PI/KF controller, as a function of frequency. The program and its capabilities are documented in Appendix B. For this study, CGTSVD was used only for the calculations involving the inverse return difference function. As shown in Section 3.2, the minimum singular value of this function is a measure of robustness with respect to a multiplicative perturbation. Analysis of this type of perturbation corresponds to classical analysis of SISO gain and phase margins [30]. Calculations were conducted with the loop broken at the control input, since those performed with the loop broken at the output required the use of a matrix pseudoinverse, and the impact on the results was not well understood. The results of the singular value analysis were then compared to simulation results in an effort to evaluate the predictive capability of this type of singular value analysis with regard to design robustness.

Neither the LQG design methodology nor the design and evaluation software cited thus far permit any "hard" constraints or nonlinearities in the system to be considered. A computer program called ODEACT (using an integration package called ODE to simulate various ACTuator models), documented in Appendix C, was written to provide the capability to evaluate the controller designs using a variety of nonlinear and higher-order actuator models, and with varying flight parameters. The program uses a "predictor-corrector" style of numerical integration scheme, implemented in a subroutine called ODE [42], to simulate the dynamics of the controlled system in conjunction with the controller being evaluated. If only higher-order dynamics had been of interest, then the linear truth model analysis capability of CGTPIV (for deterministic controllers) or PFEVAL (for stochastic controllers) could have been used. It was the need to model the actuator nonlinearities (saturation) that actually motivated the development of ODEACT.

The analysis of the baseline design through the use of ODEACT indicated that hard constraints on actuator positions and rates severely restricted the usefulness of that controller. The limits used for the simulation included a maximum deflection of 25 degrees in either direction at a maximum rate of 60 degrees per second for the horizontal tail. Limits for the flap were +20 degrees and -23 degrees at a maximum rate of 52 degrees per second [16,17]. Achieving the capability to perform under these restriction was one of the main improvements sought through the use of implicit model-following.

Addition of a higher-order model for the actuators also tended to reduce stablility in the baseline and other designs, and thus demonstrated another area in which improvement was needed. Again, the improvement was sought through the use of implicit model-following. The design model for the actuators (referred to in this thesis as the single-state actuator model) was a first-order lag with a time constant of 0.05 seconds. The transfer function between the commanded input δ_i and the achieved output δ_o was, therefore,

$$\frac{\delta_o}{\delta_i} = \frac{20}{s + 20} \qquad \text{(IV-7)}$$

The alternate actuator truth models [17] consisted either of a third-order system (also referred to herein as the three-state actuator model) with a transfer function of

$$\frac{\delta_o}{\delta_i} = \frac{(20.2)(71.4)^2}{[s+20.2][s^2 +2(.736)(71.4)s+(71.4)^2]} \qquad \text{(IV-8)}$$

or a fourth-order system (referred to as the four-state actuator model) with a transfer function of

$$\frac{\delta_o}{\delta_i} = \frac{(20.2)(144.8)(71.4)^2}{[s+20.2][s+144.8][s^2 +2(.736)(71.4)s+(71.4)^2]} \qquad \text{(IV-9)}$$

In general, initial analyses with respect to the higher-order actuator models were conducted without applying the nonlinear actuator saturation constraints, so that the effects of each could be studied separately. Additional tests were also conducted with combined nonlinear, higher-order actuators.

To evaluate the sensitivity of the regulator designs to parameter variations, ODEACT was again used. It should again be noted that this analysis could have been conducted by using various linear truth models with CGTPIV or PFEVAL; it was simply easier, in this case, to conduct the "after-design" analysis of a large number of controllers and conditions with ODEACT. The first truth model was developed by increasing the values of three of the stability derivatives (Z_α, M_α and M_q) that were used in the design model by 20%, exactly as was done in [16] to evaluate the original baseline design. Z_α is the stability derivative that relates the forces acting along the aircraft body z-axis to changes in angle of attack. M_α relates the change in pitching moment (about the body y-axis) to changes in angle of attack, and M_q relates change in pitching moment to changes in pitch rate [16]. Additional truth models were developed by using data for significantly different flight conditions (.6 mach, at 20,000 and 30,000 feet). For the case of the increased stability derivatives, the truth model dynamics matrix was

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -1.08E-3 & -2.1 & 0.994 & -0.179 & -0.295 \\ 0 & 7.2 & -0.85 & -25.3 & -5.88 \\ 0 & 0 & 0 & -20.0 & 0 \\ 0 & 0 & 0 & 0 & -20.0 \end{bmatrix} \qquad \text{(IV-10)}$$

For the flight condition at 20,000 feet, the truth model dynamics matrix was

$$\underline{A} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -2.85\text{E}-3 & -1.006 & 0.996 & -9.18\text{E}-2 & -0.1689 \\ 0 & -1.218 & -0.384 & -8.95 & -0.7786 \\ 0 & 0 & 0 & -20.0 & 0 \\ 0 & 0 & 0 & 0 & -20.0 \end{bmatrix} \qquad \text{(IV-11)}$$

and for 30,000 feet,

$$\underline{A} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -4.43\text{E}-3 & -0.666 & 0.997 & -6.06\text{E}-2 & -0.112 \\ 0 & 0.5 & -0.274 & -5.82 & -0.219 \\ 0 & 0 & 0 & -20.0 & 0 \\ 0 & 0 & 0 & 0 & -20.0 \end{bmatrix} \qquad \text{(IV-12)}$$

For these evaluations, the first-order actuator model was used, and nonlinear limits were not applied; thus, only the effect of the parameter variations from the design values was assessed.

4.4 Observations for Implicit Model-Following Design

The observations that follow emerged while working with the design model already defined, and in conjunction with the CGTPIV design software. Although some of the observed phenomena may have been a result only of the particular design problem or the design software (and certainly of the error subsequently discovered in that software, as documented in Appendix E), it is likely that much of what follows would also be true in general.

4.4.1. The regulator designs were very sensitive to the "implicit" quadratic weights placed on the output deviation rates. These weights constituted the $\underline{Q}_I$ matrix of (A-43), referred to as "QI" by the CGTPIV program output. In generating the performance index of (A-44), these weights are combined with the output, dynamics and command model dynamics matrices to define an implicit state weighting matrix. In the program CGTPIV, the result is referred to as the "QIH" matrix, which is actually the $\hat{\underline{Q}}_I$ matrix of (A-45a).

$$\hat{\underline{Q}}_I = (\underline{CA} - \underline{A}_m\underline{C})^T\underline{Q}_I(\underline{CA} - \underline{A}_m\underline{C}) \qquad \text{(A-45a)}$$

This operation distributes weights on the system states that, for the models used in this study, were much larger than would have been achieved by using, instead, an "explicit" output weighting matrix ("$\underline{Y}$" in CGTPIV and (A-39)) with a magnitude similar to that of $\underline{Q}_I$. When the command model dynamics matrix is changed, the resulting distribution changes, and the effect of the $\underline{Q}_I$ weights may be altered significantly. As a result, smaller weights (an order of magnitude or more) on the output deviation rates, compared to those which would be placed directly on the outputs or states in a standard regulator, seemed to be generally appropriate.

4.4.2. Although the implicit model-following formulation is based on matching the derivatives of the actual system's outputs to those of the command model, instead of the output values themselves, the controller does provide adequate regulation of the outputs without any need for the use of added "explicit" output weights (i.e., the $\underline{Y}$ matrix). The first reaction of the regulator in response to non-zero initial conditions was observed to be an effort to drive the outputs rapidly toward zero. This resulted in a short "rise time," defined, for the purposes of this thesis, as the time required for the pitch angle output to achieve or pass through a value within 10% of the initial excursion from the desired value of zero. Once some of the initial output deviation was nulled, the output response was observed to begin to follow the model dynamics more closely. The effect appeared to be amplified with large weights on the output rates (the $\underline{Q}_I$ matrix), and thus in the implicit state weighting matrix ($\hat{\underline{Q}}_I$). Thus, heavy weights on the output rates with a "slow" command model were not found to be effective in slowing down the initial response, characterized by the rise time of the system, as might have been anticipated. In fact, quite the opposite was observed to be true.

4.4.3. Weights placed on the input magnitudes (through the "explicit" $\underline{U}_M$ matrix of (A-39) or the "implicit" $\underline{R}_I$ matrix of (A-43)), or placed directly on the actuator states (changing the values of the $\underline{X}$ matrix of (A-40) is a CGTPIV program option), were observed to cause the initial control inputs and input rates to be increased. This resulted in increased actuator activity. The controller's strategy seemed to be to get all of the activity out of the way early, and settle down to zero values as rapidly as possible, and thus minimize the integral of the squared control/actuator magnitudes over all time. Typically, this might not be what the designer would have had in mind, nor have expected. Similarly, due to the controller structure, if the actuator states are defined as outputs of the system and modelled in the regulator command model, weights placed on the actuator output rates heavily weight the actuator states themselves. Equally important is the addition of large weights on the inputs to the actuators by the generation of the equivalent control weighting matrix. This matrix is referred to as "RIH" by the software, and is actually the $\hat{\underline{R}}_I$ matrix defined in (A-45c).

$$\hat{\underline{R}}_I = \underline{R}_I + \underline{B}^T\underline{C}^T\underline{Q}_I\underline{CB} \qquad \text{(A-45c)}$$

Thus, it was found that inclusion of the actuator states in the regulator implicit command model was not an effective way of controlling the speed of the regulator. Rather, a reduction in regulator speed, and therefore actuator activity, was found to be possible as a result of reducing the weights on the output rates ($\underline{Q}_I$) and the inputs ($\underline{U}_M$ or $\underline{R}_I$), and by modelling additional states (the pitch rate, in this case) as outputs in the regulator command model. An additional means of controlling regulator speed is discussed in Section 4.4.4, and the effectiveness of all of these techniques is shown in Chapter 5.

4.4.4. In the simple development shown in Section 2.2, the performance index for implicit model-following specifically includes quadratic weights only on the derivatives of the outputs and on the input magnitudes. But, as shown in Appendix A, the perturbation PI regulator quadratic cost also includes weights on the rate of change in control magnitudes, and this is appropriate regardless of whether or not implicit model-following is included in the formulation. Weights on the input rates were found to be required to prevent inappropriate (and ineffective) high frequency control input oscillations. In the designs attempted in this study, failure to apply some weight to the control input rates resulted in inputs which were large, and which reversed sign with each sample period. Weights on input rates were, in fact, found to be effective in achieving control over the initial speed (rise time) of the regulator as well as the extent of overshoots and control oscillations.

4.4.5. The CGTPIV program provides the user with the locations of the closed-loop regulator poles during the design evaluation. In general, this information was useful and was used extensively in the analysis of the effects of actuator dynamics on the designs. The program produces a digital controller by direct digital design methods; that is, the design procedure is accomplished in the discrete-time, or z-domain, and does not make use of approximations or mappings to allow the use of s-plane, continuous-time design techniques. There is therefore nothing to preclude the resulting controller from having poles on the negative real axis of the z-plane, which cannot be mapped into the s-domain. When this situation occurs, the algorithm, used by the program to map the controller poles into the s-plane for display to the designer, generates a pole with an imaginary part equal to pi times the sampling rate, and without a complex-conjugate mate. This occurrence should not alarm the designer; the pole information should simply be disregarded. Perhaps coincidentally, however, none of the designs in this study which exhibited this characteristic appeared to be very useful.

4.4.6. As just mentioned, the locations of the closed-loop poles of the regulator are provided by CGTPIV. By observing this information, it was possible to determine that, by use of implicit model-following design techniques, the locations of these poles can be influenced by the designer to a much greater degree than by conventional, or "explicit" regulator design. This capability was very useful, and its significance is discussed in Section 5.6.

4.4.7. The program CGTPIV assumes that, for implicit model-following, the number of inputs, outputs and command model states will all be equal. The designer may wish to increase the number of command model states in order to exercise improved control over the system. For example, in this study, the addition of the pitch rate state as a modelled output was useful in achieving control over the initial speed of the regulator. It was not possible to invent a new control to allow the dimensionality restrictions to be met, and the result was a rank deficiency in the partitioned matrix which, when inverted, produces the $\underline{\Pi}$ matrix of (II-22) and (A-21). The use of a pseudoinverse was therefore required.

The use of a matrix pseudoinverse in this situation is analogous to the use of the left inverse (a pseudoinverse) in achieving a "least-squares" solution to an over-determined set of linear equations. Rather than achieve an exact solution, which does not, in general, exist, the sum of the squared error terms is minimized. This results in a unique, minimum-norm approximation to the solution [30]. If it were desired in such a case to have the approximation be more accurate for especially critical components, a "weighted" pseudoinverse could be used in the solution [30]. The use of such a weighted pseudoinverse has not been implemented in CGTPIV, so the user has no input for specifying which elements of the resulting $\underline{\Pi}$ matrix will be determined most accurately. The situation does not in any way alter the design procedure. However, design of an acceptable controller, based on the unweighted pseudoinverse, may be more difficult, or even impossible.

In this study, regulator design was possible under these conditions, but CGT designs based on the pseudoinverse were unacceptable. Means of dealing with this problem are discussed in Section 5.5.

4.5 Summary

This chapter outlined the specific design objectives and analysis methods employed in this study. Basic insights gained while conducting designs through the use of implicit model-following were presented. In the chapter that follows, the use of implicit model-following, based on these insights, will be shown to provide a means of enhancing the ability of the designer to influence the characteristics and capabilities of the control system.

V. Analysis of Design Results

5.1 Introduction

This chapter presents the results of the design efforts of this study. The baseline controller design is defined and then analyzed in detail; this controller is shown to have little capability in tests using realistic truth models which include actuator rate and position limits or higher-order actuator dynamics. The primary concern during the actual progress of the design effort described was to produce a controller that would function well in spite of the actuator position and rate limits. The logic and design path followed in achieving that goal are presented as a means of introducing the various alternate designs; controllers based on the standard PI regulator formulation are discussed, followed by designs that used implicit model-following to try to improve on the baseline performance with respect to actuator limitations. The robustness of these designs with regard to higher-order actuator dynamics and design model parameter variations is then discussed. Finally, direct modification (via "anti-windup" methods) of the control law to compensate for nonlinearities is addressed as a complementary means of achieving design objectives. The discussion has been limited to 13 controller designs chosen from among the dozens of alternative potential controllers that were considered. The designs selected were chosen either because they were the best with regard to a particular design objective, or simply to demonstrate a point about the design technique or characteristics. An attempt has been made to keep the number of plots presented to the minimum necessary to provide insight into the relative capabilities and limitations of the different designs and the design methods used. The tables and the singular value and time-response simulation plots referred to herein have been placed together at the end of the chapter to make the text easier to read. The reader is once again referred to Appendix E for information on what aspects of the contents of this chapter were affected by the error found in the design software.

5.2 The Baseline Controller

In order to evaluate the potential of PI regulator designs developed by means of implicit model-following, a baseline, or standard for comparison, was needed. The AFTI F-16 pitch pointing CGT/PI design by Floyd [16], based on the five-state design model defined in Section 4.2, was chosen to fill this role. This design, henceforth referred to as SR-B (Standard Regulator-Baseline), was duplicated using the program CGTPIV. References [16 and 34] contain detailed explanations of this design software as well as user instructions; Appendix D documents minor changes in the program version used herein and includes a sample execution of the program. For the regulator design, quadratic weights of 200 were placed on output (pitch and flight path angle) deviations (matrix $\underline{Y}$ of (A-39)), weights of 1 each on the inputs and input rates (matrices $\underline{U}_M$ and $\underline{U}_R$ of (A-39), respectively), and an additional weight of 50 was manually inserted as the (3,3) element of the resultant state weighting matrix ($\underline{X}$ matrix of (A-40)) to penalize and thus limit the pitch rate magnitude. No "implicit" weights were used. For the CGT, a two state command model was used:

$$\underline{A}_m = \begin{bmatrix} -5 & 0 \\ 0 & -5 \end{bmatrix} \tag{V-1}$$

$$\underline{B}_m = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \tag{V-2}$$

$$\underline{C}_m = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{V-3}$$

The command model was chosen to represent the ideal output characteristics of decoupled, first-order responses for the pitch and flight path angles, which were the system outputs, to command inputs. This command model was the standard command model for all CGT designs that are discussed in this chapter. The design weights and resulting controller gains, as defined in (II-25) and (II-26), for SR-B are summarized in Table V-1.

Figure V-1 shows the response of the regulator to an initial condition of 1 degree in the pitch angle state and, therefore, in both of the outputs, recalling (IV-5) and (IV-6). So that repeated definition will not be required, all references to initial conditions of a given magnitude in the remainder of this thesis are likewise intended as an initial condition of that magnitude, in degrees, applied to the pitch angle state; both outputs therefore begin at a value of that magnitude. The time-response plot was produced using the program ODEACT (documented in Appendix C) with the linear design model as a truth model; it is identical to the terminal plot generated by CGTPIV. On this and on subsequent time-response plots, symbol "1" represents the pitch angle, symbol "2" the flight path angle, "3" is the position of the horizontal tail and "4" is the position of the trailing edge flap. With the plot held so that the title can be read, time (in seconds) is depicted on the horizontal axis and the values (in degrees) associated with the symbols are scaled separately on the vertical axis. This regulator was very fast, with a rise time of 0.12 seconds, and produced an overshoot of the final value of about 40% in the pitch angle and 15% in the angle of attack. Note, however, that the initial rate of movement of both control surfaces was far in excess of the actual rate limits of the actuators, and that the maximum deflection of the trailing edge flap exceeded the actuator position limit (these limits were defined in Section 4.3).

Figure V-2 shows the response of the CGT/PI for this design, with the same linear truth model, to a step input of 1 degree in pitch. Again, to avoid repetition, this and all subsequent references to CGT command inputs are to be interpreted as step input commands, in degrees, to the pitch angle output only. The model following was excellent. The pitch angle response approximated that of the CGT command model -- a first-order lag with a settling time of less than one second. The ideal flight path angle response would have been to remain at zero; the actual maximum error was about 0.07 degrees, with rapid regulation to zero. Both outputs achieved the desired model response steady-state values. However, the initial rate of movement of the horizontal tail surface exceeded the actuator rate limit.

The program ODEACT was next used to conduct a more realistic simulation analysis of the controller. Some of the results of that analysis are included in Table V-2. The table summarizes the results discussed in this section and also provides some data, not specifically mentioned in the text, which is useful in comparing the effects of the various perturbations on this design as well as designs to be discussed later. Figure V-3 shows the response of SR-B to an initial condition of magnitude 0.1 in both outputs, with respect to a nonlinear truth model created by adding actuator rate and position limits to the single-state (design model) actuator model. Even with this very small initial condition, the regulator showed signs of degradation due to actuator saturation; the pitch angle overshot by nearly 50%. With an initial condition of 0.5 degrees, the regulator was unstable. Figure V-4 shows the CGT/PI response to a unit step input using the nonlinear single-state actuator model. At this level of input there was very little degradation in response, but with an input magnitude of two degrees, the controller became unstable with the nonlinear truth model.

The instability of SR-B demonstrates a common effect of actuator position or rate saturation in PI controllers, referred to as "windup" [30]. When a large change in setpoint is desired, the proportional channel of the regulator can saturate the actuators; meanwhile, the integral channel begins to integrate large errors. Eventually, the integral channel will reach a level at which it can saturate the actuators by itself; the overall PI regulator command level will remain high until after the error in the output has changed sign, allowing the integral channel to "discharge". This is unlike a pure proportional gain controller which reduces its commanded control as soon as the errors come out of the saturating region. The result of saturation in the PI regulator can be large overshoots or, as seen here, instability [30]. In the case of the CGT/PI, the problem is amplified when the feedforward controls contribute to actuator saturation. While other means of compensating for this phenomenon are available [30] and will be discussed in Section 5.7, an attempt to design around the problem using implicit model-following was pursued initially.

An important question to be addressed at this point is "how large an initial condition must the regulator be able to manage in order to be considered satisfactory?" The answer depends in part on the method used to simulate the onset of the initial condition. Both ODEACT and CGTPIV use a very harsh method, which assumes that the system is totally relaxed (all states at zero) and that the states to which initial conditions are assigned are then instantaneously changed. The (perhaps more realistic) alternative to this method would be to apply the changes in states over a sequence of sample periods, transitioning from a steady-state condition to the new condition at a realistic rate that allows the controller to begin to respond during the transition period. Consider a 1 degree change in pitch at the design flight condition; causing this to occur in one sample period (.02 seconds) represents a turn rate of 50 degrees per second, which is the equivalent of a level turn at about 25 g's. Obviously, such a change in pitch through pure rotation (thus not requiring such a large translational acceleration) could occur due to gusts, or the effect duplicated by sensor errors or computer malfunction; the point is that an initial condition of 1 degree for the simulation method being used is not insignificant. The approach taken in this study was to explore the use of implicit model-following as a means to achieve regulator designs that would handle larger initial conditions than those which destabilized the baseline, SR-B; the larger the better. The goal was to examine the capability of implicit model-following design techniques. Similarly, since current flight control systems being used in the AFTI F-16 are capable of decoupled pitch-pointing angles of up to approximately 3 degrees [2], that capablity was estabished as the design objective for the CGT/PI.

Figure V-5 shows the response of SR-B to an initial condition of 1 degree with a truth model simulation employing the three-state linear actuator model described in Section 4.3; Figure V-6 shows the response using the four-state linear actuator model. Notice that the increased order of actuator dynamics caused the initial overshoot in both outputs to increase. Note also the oscillations in the control surfaces and outputs, and that the resulting degree of degradation was clearly unacceptable in the four-state simulation. Designing to avoid this problem, the effect of a "high frequency" perturbation, became another of the goals for implicit model-following.

The results with nonlinear (saturating) higher-order actuator models were predictably worse; the CGT/PI was unstable, with even a unit step input, using the three-state nonlinear actuators, due to the combined effects of actuator dynamics and nonlinearities. Again, this performance was considered unacceptable, and a candidate for improvement through implicit model-following.

The SR-B regulator was also evaluated against the additional linear truth models defined in Section 4.3 which, while of the same dimension as the design model, simulated either erroneous design model parameters or large changes in flight conditions. The first such truth model was developed using selected stability derivatives (Z_α, M_α and M_q, as described in Section 4.3) which were 20% larger than the design model values; this perturbation produced no noticeable degradation. The next truth model represented a different flight condition -- .6 mach at 20,000 feet. The regulator was degraded, but stable, as shown in Figure V-7. The final truth model represented flight conditions at .6 mach and 30,000 feet; the level of degradation was much more severe, but the regulator was still stable (Figure V-8). The differences in vertical scales of these plots is significant to note in comparing the size of the overshoots.

An analysis of the minimum singular values of the inverse return difference function of this controller was conducted using the program CGTSVD (documented in Appendix B). The singular value plots are shown in Figures V-9, V-10 and V-11. Each plot covers two decades of radian frequency, labeled (when the page is turned so that the figure title can be read) across the bottom of the plot. The symbol "1" represents the logarithm of the maximum singular value of the function at each frequency, and the symbol "2" represents the logarithm of the minimum singular value; the two values are scaled together, as indicated by the labels at the right edge of each plot. Note that the scales vary between plots. A summary of the minimum singular values (converted to actual magnitude) at various frequencies is included in Table V-3. This table also shows the smallest singular value encountered and the frequency at which it occurred ($\underline{\sigma}_{min}$ and $\omega_{\underline{\sigma}_{min}}$). As stated in Section 3.2, the magnitude of the minimum singular value of the inverse return difference function is a measure of the robustness of the controller with respect to a norm-bounded but otherwise arbitrary perturbation; in other words, we would like this number to be large, especially at high frequencies where the magnitude of uncertainty is usually large. No absolute judgment as to whether the values for SR-B are good or bad is offered at this point; their primary significance lies in comparisons with other designs.

5.3 Alternate Standard Designs

The baseline design was considered to be deficient in terms of excessive actuator commands and sensitivity to higher-order actuator dynamics, but quite robust with regard to parameter variations. Later designs with implicit model-following were able to improve in the deficient areas through reduction of output rate weightings and increased weights on control rates. In order to conduct an impartial evaluation, attempts were made to achieve the same type of improvement using the standard regulator formulation. Two of the resulting designs, SR-2 and SR-3, are summarized along with SR-B in Tables V-1, V-2 and V-3. In these designs, the output weightings were reduced and input rate weightings increased to the maximum extent possible without significantly degrading the response of the regulator with respect to the linear design model. Only improvement with regard to actuator saturation is discussed at this point; the other aspects are covered in Section 5.6. All of the simulation results discussed in this section are with respect to the truth model incorporating first-order nonlinear actuators.

The response of SR-2 with respect to initial conditions of 0.1 and 0.5 is shown, respectively, in Figures V-12 and V-13. The response was somewhat degraded at the larger initial condition, but was obviously an improvement over SR-B, which was unstable under the same conditions. SR-2 remained stable for initial conditions of up to 0.7 degrees. For a unit step input, the CGT/PI response was very similar to that of SR-B, which was depicted in Figure V-4. For an input of 2 degrees, there was very little degradation due to actuator saturation; with a 3 degree command input, the controller remained stable, although severely degraded (Figure V-14). Again, this was an improvement over the baseline design, which was unstable with an input of 2 degrees. The responses of SR-3 were very similar to those of SR-2, and are therefore not shown; the slight improvement in its performance with respect to nonlinear actuators over that of SR-2 is shown in Table V-2.

5.4 Implicit Designs with Two Model Outputs

The first designs conducted using an implicit model in the performance index of the PI regulator were based on a two-state regulator command model. Since the number of actual system outputs was equal to the number of model states, this was the normal mode for the design software being used. Some liberty was taken in using different command models in the regulator design than the one used in CGT design. Three regulator command models were used for the designs discussed in this section, all of the form

$$\underline{A}_m = \begin{bmatrix} -P & 0 \\ 0 & -P \end{bmatrix} \qquad (V-4)$$

$$\underline{B}_m = \begin{bmatrix} P & 0 \\ 0 & P \end{bmatrix} \qquad (V-5)$$

$$\underline{C}_m = \underline{I} \qquad (V-6)$$

The diagonal form for the command model dynamics matrix was chosen because decoupling of the pitch and flight path angle responses was desired. Also, it was hoped that this would contribute to the orthogonality of the eigenvectors of the resulting closed-loop system.

The first design used the same command model as the standard CGT command model, i.e., P = 5. The weightings used and gains which resulted for this design, referred to as IMF2-1 (Implicit Model-Follower, 2-state command model, design number 1), and subsequent two-state command model designs are given in Table V-4. Performance results are summarized in Table V-5; references to rise time in this section are based on operation under design conditions, as given in that table. The singular values and regulator pole locations are given in Table V-6. Once again, the discussion in this section centers on the effort to design around the actuator saturation problem; the evaluation of these designs with respect to other criteria is reserved for Section 5.6. All of the simulations discussed in this section were conducted using the single-state nonlinear actuator model.

IMF2-1 was not a great improvement over the standard regulators. Even though the weightings on the output rate deviations were less than the corresponding weights on output deviations for the earlier standard regulator designs, these weights were transformed by (A-45a) to produce very high state weightings -- greater in some cases by an order of magnitude over the standard regulator designs. The result was, again, short rise time. Figures V-15 and V-16 show the response of IMF2-1 to initial conditions of 0.1 and 0.5 degrees, respectively. As with SR-2 and 3, there was little degradation at the smaller initial condition; the larger initial condition caused greater degradation, but not instability. The CGT/PI response to a unit step input was much the same as the standard formulation designs. The response with a commanded input of 3 degrees was an improvement, however, as shown by comparison of Figure V-17 to Figure V-14.

IMF2-2 was an attempt to slow down the initial response of the regulator by changing the command model to P = 2. This path was chosen to limit the rate of increase in control inputs to the actuators, and thus reduce the effects of the discrepancy between the actuator rate capability of the nonlinear actuator model and that of the linear design model. It was hoped that this would reduce the amount of degradation due to actuator rate saturation. As shown in subsequent designs, slowing down the regulator in such a manner was effective, and tended to reduce the effects of the nonlinear actuator position limits, as well. However, the quadratic weightings used in IMF2-2 were the same as for IMF2-1, so the state weightings were still quite high, and there was not a significant difference in the speed of the regulator or in the magnitude or rate of control surface movement (with a linear truth model) from that of the previous design. The response to a small initial condition (0.1) is shown in Figure V-18, with the regulator displaying less of a tendency to overshoot than IMF2-1, as seen by comparison with Figure V-15. The regulator remained stable for initial conditions of up to 0.9 degrees. The CGT/PI response to a unit step input was similar to previous designs, but the pitch angle did overshoot the commanded value by a small amount, as shown in Figure V-19. However, unlike the response of IMF2-1 shown in Figure V-17, there was no increase in the percentage of overshoot of the CGT/PI pitch angle when the command was increased to 3 degrees.

IMF2-3 incorporated both the slower command model and reduced quadratic weights on the output rates, as shown in Table V-4. An increase was also made in the weights placed on the input rates. As a result, significant differences became apparent between this and previous designs. Because of the changes in quadratic weights, the initial response of the regulator was slower, with a rise time of 0.16 seconds, and produced smaller control inputs. For this reason it was able to remain stable with an initial condition of 1.1 degrees, although it was certainly degraded in performance at that level. Figure V-20 shows the response to an initial condition of 1 degree. The CGT/PI response to a unit step input was similar to that of IMF2-2 (which was shown in Figure V-19); for an input of 3 degrees, the overshoot was equal in magnitude to that of IMF2-1 (which was shown in Figure V-17), but subsequent recovery was slower, as shown in Figure V-21.

IMF2-4 incorporated even lower weights on the output rates and higher weights on the input rates, with a continued performance improvement relative to actuator saturation. Rise time increased to 0.18 seconds, and as shown in Figure V-22, the response to an initial condition of 1 degree was better than with any previous designs. The regulator remained stable with initial conditions of up to 1.5 degrees. The CGT/PI response to a unit step input was similar to the earlier designs. As shown in Figure V-23, however, the pitch angle overshoot in the CGT/PI response with a large (3 degree) input was greater, and recovery more prolonged, due to the slower regulation of errors.

IMF2-5 used weights similar to those of the previous design and used an even slower command model, with P = 1.5, as a test of how an extremely slow regulator would perform. The response of the regulator to a small initial condition (0.1) is shown in Figure V-24; its rise time was the same as that of IMF2-4 (Table V-5) but the regulation of the pitch angle after the initial overshoot was slower. This regulator was stable with initial conditions of up to 1.7 degrees, although with an initial condition of 1 degree, the overshoot in pitch angle was greater (170% versus 130%) than for IMF2-4. The CGT/PI response to a unit step input is shown in Figure V-25. In this case, the level of input was such that actuator saturation was not yet a factor, but the overshoot was larger, and the subsequent correction was slower, than in previous designs. With larger commands, and the onset of saturation, the effect was amplified. Since the regulator for this design represented no real improvement over IMF2-4 and the CGT/PI was degraded, IMF2-5 was felt to represent the practical limit to which the regulator could be slowed, by the use of a slow regulator command model, without creating an unacceptable design.

It was seen in this section that implicit model-following could be used to enhance the capability of the controller to perform with nonlinear actuators subject to position and rate saturation. This was accomplished by slowing down the initial response of the regulator, as characterized by rise time. The desired effect was achieved by using lower quadratic weights on the output rates in conjunction with higher weights on the input rates, as well as a slower regulator command model. As the rise time of the successive designs increased, the ability of the regulators to perform well and remain stable with larger initial conditions improved, as summarized in Table V-5. However, as the regulator speed decreased, the precision of the CGT/PI response to command inputs was degraded slightly.

5.5 Implicit Designs with Three Model Outputs

Designs using a three state command model for the regulator were undertaken in an attempt to achieve more control over the system's closed-loop characteristics. In any design, the degree to which full eigenstructure assignment can be achieved is ultimately limited by the number of outputs and controls available. If the controlled system has r independent controls and p independent outputs available for feedback, then at most max(r,p) eigenvalues may be assigned and min(r,p) entries of max(r,p) closed-loop eigenvectors may be assigned [1]. Increasing the number of outputs in the design model was not a problem in this case since all states were assumed available, but there was no way to increase the number of controls. The pitch rate was chosen as the third output to provide a means of directly controlling the speed of the response, and a corresponding third state was added to the regulator and CGT command models, which became

$$\underline{A}_m = \begin{bmatrix} -P_1 & 0 & 0 \\ 0 & -P_2 & 0 \\ 0 & 0 & -P_3 \end{bmatrix} \qquad \text{(V-7)}$$

$$\underline{B}_m = \begin{bmatrix} P_1 & 0 & 0 \\ 0 & P_2 & 0 \\ 0 & 0 & P_3 \end{bmatrix} \qquad \text{(V-8)}$$

$$\underline{C}_m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \text{ (or 0 for the CGT)} \end{bmatrix} \qquad \text{(V-9)}$$

Again, the diagonal form for the regulator command dynamics matrix was initially chosen to attempt to enhance the orthogonality of the closed-loop system eigenvectors. For the CGT command model, P_3 was set to zero, as was the (3,3) element of the model output matrix, so that the effective model was the same as for the earlier designs, but with the CGTPIV dimensionality restrictions met.

As mentioned in Section 4.4.7, the CGTPIV software requires that the number of controls and outputs be equal. In conducting a design when this is not really the case, either the design model control or output matrix will be rank-defective. The entire design result is thereby based on an unweighted matrix pseudoinverse (the program output announces this by stating that the $\underline{\Pi}$ matrix is rank-defective). If the number of controls is less than the number of outputs, then a true solution will only be achieved in a least-squares sense. The designer can still exercise a great deal of control over the performance of the resulting regulator by iteratively changing the weightings used in its design; there is no corresponding means of compensation, however, to influence the design of the CGT. The result in this case was that the regulator designs were relatively successful in that they produced acceptable performance with more conservative control surface commands; but the CGT designs were all unsatisfactory in that the feedforward gains were too high, resulting in excessive commands to the actuators. All of the CGT designs conducted with the rank-defective $\underline{\Pi}$ matrix exhibited performance degradation, with even a unit step input, when evaluated using the nonlinear single-state actuator model. This was true even though the CGT command model used employed $P_3 = 0$ and $\underline{C}(3,3) = 0$, thus preventing any feedforward control from being generated based on the added model state. Subsequent study showed that the regulators discussed in this section could, however, be used in conjunction with the CGT designs of the preceding section, provided the regulator response was reasonably similar to that of the regulator upon which the CGT design was based. Controllers defined in such a manner are discussed at the end of this section.

The CGT/PI designs discussed in this section are summarized in Tables V-7, V-8 and V-9. Note that, unlike previous performance summaries, Table V-8 does not indicate the level of initial condition required to produce a 100% overshoot; this is because the onset of actuator degradation due to saturation was always within 0.5 degrees of the maximum stable initial condition. Once again, the presentation at this point will center on the attempt to alleviate the actuator saturation problem; all simulations are with respect to the single-state nonlinear actuator model.

Design IMF3-1 was based on a regulator command model with $P_1 = P_2 = 5$, $P_3 = 10$, and what would seem to be a light weight on the pitch rate's rate deviations (0.1). These initial values of P_1 and P_2 were chosen to correspond to the regulator command model used for the first designs of the earlier IMF2 class. The larger value for P_3 was found to be needed to achieve a response with reasonable speed characteristics. Likewise, the small weight on the pitch rate derivative was chosen as a result of trial-and-error attempts using a larger value, which produced an excessively slow regulator. This regulator's response to an initial condition of 1 degree in pitch and flight path angle is shown in Figure V-26, and demonstrates just how great an effect the additional model state had on the speed of the regulator; the rise time was 0.62 seconds, as compared to 0.18 seconds for the slowest regulator in the IMF2 class. There was no noticeable degradation due to actuator saturation for initial conditions up to 6 degrees; and stability was maintained for initial conditions up to 6.5 degrees. However, it was felt that the response of this regulator was too slow for adequate performance in a fighter aircraft flight control system.

IMF3-2 employed a faster command model ($P_1 = P_2 = 10$, $P_3 = 15$) and an even lower weight on the pitch rate's rate deviations (0.01) in an effort to achieve a faster response. This was achieved without any degradation due to actuator saturation, at least with an initial condition of 1 degree, as shown in Figure V-27. The rise time was reduced to 0.32 seconds; the price of the improvement was that this regulator could withstand smaller initial conditions than IMF3-1 -- a maximum of 2.5 degrees.

IMF3-3 differed from its predecessor by virtue of a regulator command dynamics matrix with an off-diagonal term (Table V-7) and a decrease in the pitch angle output rate weighting. The purpose of the non-diagonal command dynamics matrix was to investigate the effect of recognizing and allowing a degree of coupling of the pitch state with its derivative. Full recognition of the physical relationship between the pitch and pitch rate states could have been realized by additionally setting $P_1 = 0$, but this would have represented a radical departure from the stated design philosophy of attempting to orthogonalize the closed-loop system eigenvectors through the use of implicit model-following with a diagonal regulator command dynamics matrix. At this point, only a small departure was desired, since the effect on performance and robustness had not been determined. There was nothing inherently wrong with the response of IMF3-3, but many other structures for the command dynamics matrix were also possible, each representing different types and degrees of coupling among the outputs. Since no insight was readily available to indicate which of such structures (other than the use of the diagonal form) might produce beneficial results, further investigation with the non-diagonal regulator command dynamics matrix was felt to be beyond the scope of this study. However, such an investigation could be a fruitful area for future research.

The design using the non-diagonal command dynamics matrix was originally attempted using the same output rate weightings as had been used for IMF3-2. The pitch angle overshoot for the resulting regulator's response to initial conditions was 25%, compared to 5% for IMF3-2; and the response was poorly damped, resulting in repeated overshoots of the pitch angle steady-state value. To alleviate this condition, the weight on the pitch angle output rate was reduced, as shown in Table V-7, to produce the final IMF3-3 design. Its response to an initial condition of 1 degree is shown in Figure V-28; the response was slightly slower than that of IMF3-2, shown in Figure V-27, and the pitch angle overshoot was about 15%. While this overshoot was greater than for the other IMF3 designs, it was less than for any of the standard regulator and IMF2 designs.

Many attempts were made to design a regulator that would work well with a CGT design based on the rank-defective $\underline{\Pi}$ matrix. Initially, it was thought that the CGT might be "fighting" the slower regulator response, thus causing the excessively large control inputs through the CGT; however, speeding up the regulator through faster command models and modified weightings in the performance index had virtually no effect on the level of control input produced by the CGT/PI. Attempts were then made to slow the regulator down even further, with the aim of limiting the actuator commands. By the time any positive results were achieved by this approach, the regulator performance was considerably degraded; even then, the CGT was still unsatisfactory. Two of the "slow" regulators which resulted from these design attempts were IMF3-4 and IMF3-5. No alteration of the CGT command model was attempted. CGT control inputs could have undoubtedly been reduced by using a slower CGT command model (with smaller P values), but such a change would have also slowed the CGT/PI response to command inputs. Such a result was not desired.

IMF3-4 reverted to the slower regulator command model used for IMF3-1, and greater penalty was placed on the input rates. This slowed the regulator down, when viewed in terms of both rise and peak times, and reduced regulator control inputs considerably. The response to initial conditions is shown in Figure V-29; the shape of the pitch response was somewhat unusual due to the simultaneous reversal in motion of both control surfaces.

IMF3-5 was slowed even further by lower output rate weights and lower input weights. The results were a continuation of the trends established by the previous design. The irregular response of the pitch angle in recovering from initial conditions is shown in Figure V-30; because of this, the controller would be an unlikely candidate for final implementation.

As mentioned earlier, none of the CGT designs conducted with the rank-defective $\underline{\Pi}$ matrix were suitable, due to excessive feedforward control magnitudes. However, several of the earlier CGT designs based on the two-state command model worked quite well with regulators discussed in this section. Since the AFTI F-16 is an unstable plant, an open-loop CGT design is not feasible. To design a CGT, a stabilizing regulator design must first be accomplished; then the CGT design can be based on the stable plant/regulator combination. Thus, a CGT based on the two-state command model is also based on a specific regulator (of the IMF2 class, in this case).

Satisfactory combinations of previous CGT designs and IMF3 regulators were identified by trial-and-error, and generally resulted when the initial response dynamics (i.e., rise time) of the IMF3 regulator were similar to those of the IMF2 regulator upon which the CGT design was based. As an example, the CGT designed for controller IMF2-5, which was the slowest regulator of its class (Table V-5), worked well with regulator IMF3-2, which was the fastest of the slower class (Table V-8). The response of this hybrid CGT/PI to a large step input of 3 degrees is shown in Figure V-31, which indicates very little degradation due to actuator saturation. An example of a mis-matched design occurred by using the CGT of IMF2-2 (which, as shown in Table V-5, was considerably faster than IMF2-5) with the same regulator, as shown in Figure V-32.

It was seen in this section that the addition of the pitch rate state as a modelled output provided a considerable increase in the degree of control over the initial response speed of the regulator. As a result, the regulators were able to withstand much larger initial conditions than earlier designs, both in terms of stability and performance degradation due to actuator saturation. While CGT designs conducted with a rank-defective $\underline{\Pi}$ matrix were unsuitable, the employment of hybrid designs was shown to be feasible.

5.6 Robustness Analysis

In the preceding sections, the use of implicit model-following in PI regulator design was shown to be an effective means of limiting the magnitude and rate of controller inputs. This, in turn, helped to prevent performance degradation and instability due to actuator saturation. In this section, the robustness of the various designs previously introduced is characterized further. Each controller was evaluated against linear truth models which differed from the design model in that third- and fourth-order actuator dynamics were simulated. Each controller was also evaluated against linear truth models with design model actuators, but with other system parameters which varied from those of the design model. The results of these evaluations were compared to information obtained through singular value analysis and study of the regulator pole locations.

The baseline regulator was previously shown to be seriously degraded by the addition of higher-order actuator dynamics (Figures V-5 and V-6, and Table V-2). Since the three- and four-state actuator models had complex poles at a natural frequency of 71.4 radians per second, it was thought likely that the resulting perturbation could be characterized as having a maximum norm at relatively high frequency, say between 10 and 100 radians per second. Thus, the designs for which the minimum singular values of the inverse return difference were largest over this range were expected to perform better with these truth models. As noted in Section 3.2, however, the conservatism of this type of analysis can vary considerably, primarily as a function of the vector direction of the perturbation. To complement the analysis of singular values with techniques of a more classical flavor, the locations of the closed-loop regulator poles for each design were noted. The higher-order actuator model poles were relatively far from the origin, and had the effect of pushing the complex poles of the design model-regulator combination closer to the imaginary axis. The destabilizing result is analogous to the addition of such poles to the open-loop transfer function of a SISO control system [8].

The singular values and pole locations for the standard regulators are summarized in Table V-3. SR-2 had slightly larger singular values than the baseline design over most of the frequency range of interest, although the difference was not large. The complex regulator poles were located slightly further from the imaginary axis than those of the baseline. The response of SR-2 to initial conditions with the linear four-state actuators was slightly better than that of the baseline, as shown in Figure V-33. SR-3 also had a small singular value advantage over SR-B. Its complex regulator poles were closer to the origin in both horizontal and vertical directions, but were also further from the added actuator poles. This regulator suffered much less degradation with the higher-order dynamics than the other two, as shown in Figure V-34. It was noted with these and with other intermediate designs that the pole locations varied with the changes in quadratic weightings used to design the regulator, but that the pattern of motion was not very consistent. With each change, some poles moved in one direction, some in another. More systematic motion occurred with implicit model-following techniques, as will be seen in subsequent paragraphs.

The singular values and pole locations for the IMF2 class of controllers are summarized in Table V-6. The singular values of IMF2-1 and IMF2-2 were nearly identical; both were slightly better than the baseline at 10 radians per second, and worse at 100 radians per second. The complex pole locations, two sets in this case, were also nearly identical. The location of the largest complex poles (nearest the added actuator poles and thus most vulnerable to their effects) was better than the baseline in that they were further from both the imaginary axis and the added actuator poles. The resulting performance with the four-state actuator model was similar for these two designs. Figure V-35 shows the response of IMF2-1, which was the more oscillatory of the two. They were better than SR-B and SR-2, but slightly worse than SR-3 in terms of oscillation. The singular values of IMF2-3 were consistently equal to or better than the baseline over the frequency range of interest. The complex poles were much closer to the origin, and thus further from the added actuator poles. As a result, performance with the four-state actuator model was much better than earlier designs, as shown in Figure V-36. The same trends exhibited by IMF2-3 were continued to an even greater extent by IMF2-4 and IMF-5; these regulators suffered only relatively small increases in overshoot when subjected to the actuator dynamics, as shown for IMF2-4 in Figure V-37.

Refer again to Tables V-3 and V-6. Note that the use of implicit model-following produced a greater range of high frequency singular values than did the standard regulator formulation. This indicates that the implicit method provides the designer an added degree of control over the characteristics of the design. The same conclusion is supported by movement of the regulator poles in the implicit model-following designs. The value of P selected for the regulator command model very nearly determined the location of the regulator poles nearest the origin; the higher the weights on the output rate deviations, the more precisely this was true. As weights on the output rate deviations were decreased, the remaining poles moved consistently toward the origin (for a constant P); the same trend was followed as the weights on the input rates were increased. Both the degree and consistency of control available over the regulator pole locations is significant in providing the designer with a method of designing around a specific problem, as was the case with the higher-order actuator dynamics.

Singular values and pole locations for the IMF3 class of controllers are summarized in Table V-9. The singular values for IMF3-1 were smaller than the baseline regulator over most of the desired frequency range. The radical location of the large complex poles for this regulator was sufficient to cause it to be just unstable when tested with the four-state actuator model, as shown in Figure V-38. IMF3-2 had only slightly larger singular values than IMF3-1, but its pole locations were significantly different. The large complex poles were moved much nearer the origin by the decrease in weight on the derivative of the pitch rate. As a result, the regulator was barely affected by the dynamics of the four-state actuator model, as shown in Figure V-39. The singular values and pole locations for IMF3-3 were similar to those of IMF3-2; it was not degraded by the actuator dynamics. The singular values listed for IMF3-4 and IMF3-5, if considered alone, would have predicted excellent performance against a high frequency perturbation. The relative proximity of the complex poles of these regulators to the imaginary axis was, however, enough to cause serious performance problems, as shown in Figures V-40 and V-41. In this case, consideration of the exact perturbation, i.e., added poles at a known location, was far more informative than the analysis of singular values.

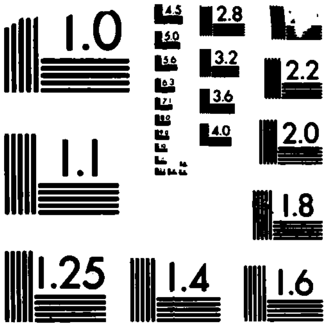
It was concluded earlier that the baseline design was quite robust with regard to parameter variations to the design model. Each of the other controllers was evaluated by use of the same alternate linear truth models as was SR-B, in an attempt to assess their robustness further. It was felt that these parameter variations could be characterized as perturbations which would have an effect over a wide range of frequencies. That being the case, it was anticipated that the overall smallest value encountered as a minimum singular value for the inverse return difference function would be the best predictor of robustness, since it represented the point of greatest vulnerability. Reference should be made to the singular value summaries in Tables V-3, V-6 and V-9 during the discussion that follows.

The first alternate linear truth model used was the one developed by using selected stability derivative values 20% larger than in the design model. This model represented a fairly large modelling error, yet none of the designs were noticeably degraded. This indicated that all of the designs were reasonably robust against this type of error, one of a type which might be considered reasonably likely to occur physically.

The second truth model was one representing flight dynamics at .6 mach and 20,000 feet. For a control system which would normally employ gain scheduling to optimize control over a wide range of flight conditions, a perturbation of this size would probably physically only occur due to a catastrophic sensor or computer error; it is a harsh test of the controller. The response of the two standard regulators, SR-2 and SR-3, with this truth model was very similar to that of SR-B, which was shown earlier in Figure V-7. The responses of the IMF2 class of regulators were all slightly degraded in comparison to those of the standard regulators, with somewhat larger initial overshoots and oscillations. It was difficult to rank them objectively with a four-second simulation, although the performance of IMF2-4, shown in Figure V-42, was typical of the group, and the data in Table V-5 provides some insight into the rate with which the oscillations died out. The IMF3 class of regulators exhibited a wider range of performance than the other groups. IMF3-1 was hardly even degraded, as shown in Figure V-43. IMF3-2 and IMF3-3, on the other hand, were barely unstable under these conditions, while IMF3-4 and IMF3-5 were just barely stable.

The final truth model represented flight conditions at .6 mach and 30,000 feet. All of the standard regulators remained stable, with performance similar to that already shown for SR-B in Figure V-8. The IMF2 controllers all exhibited slowly divergent oscillations except for IMF2-2, which was barely stable; the performance within the group was, once again, fairly consistent. Again, a wider range of performance was seen with the IMF3 group. IMF3-1 was stable, and in fact did rather well, as shown in Figure V-44. IMF3-2 and IMF3-3 diverged very rapidly, while IMF3-4 and IMF3-5 were just unstable for this condition. The relative results of the 30,000 foot evaluation were, therefore,

AD-A138 309   ROBUST MULTIVARIABLE CONTROLLER DESIGN VIA IMPLICIT   2/4
MODEL-FOLLOWING METHODS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..   W G MILLER
UNCLASSIFIED   DEC 83 AFIT/GE/EE/83D-48   F/G 14/2   NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

consistent with the one at 20,000 feet, but the effects were more severe.

In comparing the results of the parameter variation tests for the standard regulators to those of the IMF2 class, the overall minimum singular value encountered seemed to be a reasonably valid predictor of robustness, as had been anticipated. The overall minimum singular values for all of the standard regulators (shown in Table V-3) were better than those for any of the IMF2 controllers (Table V-6); and unlike the IMF2 class, none of the standard regulators were destabilized by any of the parameter variations. Within the IMF2 class, however, the differences in the minimum singular values were fairly small, as were the differences in the observed robustness characteristics of the designs; the robustness differences were not predictable based on the minimum singular values, as shown by comparison of Table V-5 and Table V-6.

The results were even less clear with the IMF3 class, where the range of minimum singular values was significantly greater, as was the range of robustness qualities. The minimum singular value for IMF3-1 was the greatest for any design in either implicit class, and about average when compared to the standard regulators; its good performance in these tests was therefore not surprising. But with the other IMF3 regulators, those based on the faster command model performed better than those based on the slower command model, regardless of their minimum singular values. This result is a reminder that, for this type of perturbation (an additive change in the nominal dynamics matrix), the perturbation is a function not only of the change to the dynamics matrix, but of the control law itself. In fact, for a change $\Delta \underline{A}$ to the nominal dynamics, and using $\underline{K}(s)$ to represent a continuous-time control law, the norm of an equivalent multiplicative perturbation as presented in Section 3.3 could easily be calculated.

Since

$$\underline{KG}' = (\underline{I} + \underline{L})\ \underline{KG} \qquad \text{(III-7)}$$

and here

$$\underline{G}(s) = (s\underline{I} - \underline{A})^{-1}\underline{B} \qquad \text{(V-10)}$$

$$\underline{G}'(s) = (s\underline{I} - [\underline{A} + \Delta\underline{A}])^{-1}\underline{B} \qquad \text{(V-11)}$$

we have

$$\underline{K}(s)\underline{G}'(s) = [\underline{I} + \underline{L}(s)]\underline{K}(s)\underline{G}(s) \qquad \text{(V-12)}$$

$$\underline{K}(s)[s\underline{I} - (\underline{A} + \Delta\underline{A})]^{-1}\underline{B} = [\underline{I} + \underline{L}(s)]\underline{K}(s)[s\underline{I} - \underline{A}]^{-1}\underline{B} \qquad \text{(V-13)}$$

$$[\underline{I} + \underline{L}(s)] = \left\{\underline{K}(s)[s\underline{I} - (\underline{A} + \Delta\underline{A})]^{-1}\underline{B}\right\}\left\{\underline{K}(s)[s\underline{I} - \underline{A}]^{-1}\underline{B}\right\}^{-1} \qquad \text{(V-14)}$$

Therefore,

$$\bar{\sigma}\,[\underline{I} + \underline{L}(s)] = \bar{\sigma}\left\{\underline{K}(s)[s\underline{I} - (\underline{A} + \Delta\underline{A})]^{-1}\underline{B}\right\}\left\{\underline{K}(s)[s\underline{I} - \underline{A}]^{-1}\underline{B}\right\}^{-1} \qquad \text{(V-16)}$$

In other words, in comparing two different controllers, the difference in minimum singular values could be easily overshadowed by the effect of the differing control laws on the magnitude and, perhaps more importantly, the direction of the perturbation. This may explain why a fairly radical change in the basis for the controller design, such as a different regulator command model, could make predictions of robustness based on relative sizes of singular values less meaningful.

In general, the regulator designs which exhibited relative immunity to the effects of higher order actuator dynamics were those most vulnerable to the low frequency perturbations. The clearest example of this was within the IMF3 group of controllers. Apparently, a trade-off decision is required as to which type of uncertainty should be most heavily guarded against. In the case of the AFTI F-16, the "uncertainties" of nonlinearities and higher-order actuator dynamics are, in fact, a reality; the controller must work despite them. The implicit model-following designs, which fulfilled this requirement, were also robust for a reasonable and realistic level of modelling error.

In summary, singular value analysis was of only limited value in predicting the robustness and sensitivity of the designs with regard to parameter variations and higher-order dynamics. The variablity of the singular value measure's conservatism caused it to be misleading in some cases. Regulator pole analysis and actual simulation provided reliable information and a significant degree of physical insight. The ability of the implicit model-following formulation to affect the location of the regulator poles in a systematic manner was found to be very helpful in avoiding problems associated with higher-order dynamics, but this was generally at the expense of increased sensitivity to parameter variations.

5.7 Anti-Windup Compensation

It has been shown that implicit model-following can be used to alleviate the windup problems that occur in PI regulators when nonlinear, or rate- and position-limited, actuators are encountered. The regulators in the IMF3 group were especially effective in this regard; all were capable of handling at least a 2.5 degree initial condition, without performance degradation, when tested against the nonlinear single-state actuator model. The hybrid CGT/PI using the regulator from IMF3-2 and the CGT from IMF2-5 was shown to provide good pitch-pointing of up to 3 degrees with the same truth model.

When the effects of higher-order actuator dynamics were subsequently considered in addition to the rate and position limits, the capabilities of these designs were decreased somewhat, as expected. However, in additional tests it was found that IMF3-2 and IMF3-3 were still stable and performed without degradation with initial conditions of 1 and 2 degrees, respectively, using a full, nonlinear four-state actuator model. A hybrid CGT/PI was also found which was capable of good pitch-pointing control of up to 3 degrees with the full, nonlinear four-state actuator truth model, as shown in Figure V-45. This hybrid used the regulator of IMF3-3 and a CGT based on a regulator design which, while of the IMF2 class, and similar in its definition and performance characteristics to IMF2-4, was not among those chosen to be discussed in this thesis. The CGT gains for the hybrid were

$$\underline{K}_{xu} = \begin{bmatrix} -3.38 & -8.406 \\ -1.695 & 25.24 \end{bmatrix} \qquad \text{(V-16a)}$$

and

$$\underline{K}_{xm} = \begin{bmatrix} 2.747 & -14.05 \\ -9.724 & 22.94 \end{bmatrix} \qquad \text{(V-16b)}$$

While the use of a controller that is designed specifically to remain within the linear operational range of the actuators is a worthwhile goal, it may not always be possible to implement. Even if such a design were possible, it would be wise to implement some form of a safeguard that would enhance the stability of the controller in the event that actual conditions exceeded the linear range for which the controller was designed. Since some form of additional anti-windup compensation should thus inevitably be used with any practical controller implementation of the type addressed in this study, it seemed imperative that its effect on CGT/PI designs be considered.

The form of anti-windup compensation investigated in this study involved a simple modification to the control law [30], which was implemented as an additional user option in ODEACT. At each sample period, when the new control inputs are calculated, a check is made to ensure that those control inputs are not of a magnitude that would command the rate or position limits of the actuators to be exceeded. If the calculated control levels are excessive, they are reduced to the maximum allowable level. With that restriction satisfied, windup does not occur, since the actuators do not saturate. Actually, several variations in the method of checking and limiting the controls were tried; the most successful implementation is discussed here.

Consider only the horizontal tail actuator, which was position limited to ± 25 degrees, and rate limited to ±60 degrees per second.

Even though the fourth-order actuator dynamics model is more correct, the design model, of a simple first-order lag, is a good approximation and much more tractable for the implementation of control-law compensation. The equivalent discrete-time model for the actuator dynamics is

$$x(t_{i+1}) = \Phi(t_{i+1},t_i)x(t_i) + \int_{t_i}^{t_{i+1}} \Phi(t_{i+1},\tau)Bd\tau u(t_i) \qquad \text{(V-17)}$$

In this case, the sampling period is .02 seconds, and the first-order lag coefficient is 20.0, so

$$x(t_{i+1}) = 0.67x(t_i) + 0.33u(t_i) \qquad \text{(V-18)}$$

Since $x(t_{i+1})$ cannot exceed ± 25 degrees, this yields a bound on $u(t_i)$, in degrees, as

$$u(t_i) \leq 75 - 2x(t_i), \text{ for } x(t_i) \geq 0 \qquad \text{(V-19a)}$$

or

$$u(t_i) \geq -75 - 2x(t_i), \text{ for } x(t_i) < 0 \qquad \text{(V-19b)}$$

Since the difference between $x(t_{i+1})$ and $x(t_i)$ cannot exceed 1.2 degrees (60 degrees per second times the sample period),

$$x(t_{i+1}) - x(t_i) = 0.67x(t_i) + 0.33u(t_i) - x(t_i) \leq 1.2$$

or

$$u(t_i) \leq 3.6 + x(t_i), \text{ for } x(t_i) \geq 0 \qquad \text{(V-20a)}$$

and

$$u(t_1) \geq -3.6 + x(t_1), \text{ for } x(t_1) < 0 \qquad \text{(V-20b)}$$

An identical development was used to calculate the maximum controls for the trailing edge flap, except that the appropriate limits for that actuator were used.

With this simple modification to the control law, it was found that the IMF3-2 and IMF3-3 regulators could withstand at least an additional 1 degree of initial conditions against the full, nonlinear four-state actuator model, without instability being induced. The IMF2 regulators showed a much greater improvement with anti-windup compensation than did the IMF3 regulators. The response of IMF2-4 to an initial condition of 10 degrees using the full nonlinear four-state actuator model is shown in Figure V-46; the quality of this response was excellent.

With the same actuator model, the IMF2-4 CGT/PI response to a step input of 3 degrees is shown in Figure V-47; while the performance is acceptable, it should be compared to that of the hybrid CGT/PI's under the same conditions. The response of the hybrid controller whose response without compensation was shown in Figure V-45 is shown with anti-windup compensation in Figure V-48. Since the uncompensated response was quite good, the improvement due to compensation was minimal, primarily seen in the smaller extent of the actuator overshoots and the smaller maximum flight path angle deviation. However, compared to the IMF2-4 response of Figure V-47, this hybrid design produced better model-following in the pitch angle, plus a smaller and more rapidly corrected deviation in the flight path angle. The other hybrid CGT/PI, constructed from the elements of IMF3-2 and IMF2-5, was improved considerably through the use of anti-windup compensation. Without such compensation, it was unstable with a 3 degree step input when evaluated against the nonlinear four-state actuator model; with compensation, its response is shown in Figure V-49. By all standards of comparison with Figure V-48, the hybrid based on IMF3-2 was nearly equal to or better than the one based on IMF3-3, which had performed better in the uncompensated mode.

The baseline regulator also showed significant improvement when anti-windup compensation was used; this was to be expected, since it had the most to gain from such compensation. However, as with some of the implicit designs, the controller was still not useful, since it was still degraded by the effects of the higher-order actuator dynamics.

In summary, the employment of anti-windup compensation has been shown to enhance the capabilities of the controller designs. When used with implicit model-following regulators that were immune to the effects of higher-order actuator dynamics, both stability and performance improvements were realized, resulting in potentially very useful designs.

Table V-1. Definition of Designs Using Standard PI Regulator Formulation

Design	Quadratic Weights*	Gains**	
		Regulator	CGT
SR-B	$\underline{Y}$=diag(200,200) $\underline{U}_M$=diag(1,1) $\underline{U}_R$=diag(1,1) $\underline{X}$(3,3)=50	$\underline{K}_x$: $\begin{matrix} -38.87 & 19.69 & -1.687 & 1.282 & 5.7E-2 \\ 78.07 & -83.61 & 0.4227 & 5.5E-2 & 1.045 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} -9.487 & -14.92 \\ -18.44 & 61.13 \end{matrix}$
		$\underline{K}_z$: $\begin{matrix} 7.0E-3 & -0.8716 \\ -2.929 & 3.068 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} -5.908 & -8.554 \\ -1.059 & 36.43 \end{matrix}$
SR-2	$\underline{Y}$=diag(50,50) $\underline{U}_M$=diag(1,1) $\underline{U}_R$=diag(.5,2) $\underline{X}$(3,3)=50	$\underline{K}_x$: $\begin{matrix} -45.49 & 29.95 & -2.08 & 1.827 & 5.37E-2 \\ 45.33 & -49.79 & 0.2143 & 3.66E-2 & 0.6242 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} -3.407 & -23.64 \\ -13.74 & 32.68 \end{matrix}$
		$\underline{K}_z$: $\begin{matrix} 0.8258 & -1.207 \\ -1.726 & 1.792 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} -7.317 & -11.12 \\ -1.822 & 28.21 \end{matrix}$
SR-3	$\underline{Y}$=diag(50,50) $\underline{U}_M$=diag(1,1) $\underline{U}_R$=diag(1,3) $\underline{X}$(3,3)=25	$\underline{K}_x$: $\begin{matrix} -30.83 & 18.41 & -1.457 & 1.213 & 5.48E-2 \\ 33.97 & -37.41 & 0.1372 & 4.5E-2 & 0.4717 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} -3.194 & -13.8 \\ -11.39 & 22.28 \end{matrix}$
		$\underline{K}_z$: $\begin{matrix} 0.3444 & -0.7674 \\ -1.304 & 1.377 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} -4.827 & -8.284 \\ -2.091 & 25.16 \end{matrix}$

* "diag(•,•,•)" means a diagonal matrix consisting of the enclosed values.

** See Equations (II-25) and (II-26) and Appendix E.

Table V-2. Summary of Performance Analysis for Standard PI Regulator Designs

Condition and Measure of Performance	SR-B	SR-2	SR-3
Percentage overshoot in pitch angle (truth model=linear design model)	40 (Fig V-1)	42	55
Rise time/peak time (seconds) for pitch angle (truth model=linear design model)	0.12/0.22 (Fig V-1)	0.10/0.20	0.12/0.22
Pitch angle initial condition (degrees) to cause 100% overshoot (nonlinear actuators *)	0.4^-	0.4^+	0.5
Maximum pitch angle initial condition (degrees) for stability (nonlinear actuators *)	0.5^-	0.7^+	0.8
Maximum CGT pitch input (degrees) for stability (nonlinear actuators *)	1.5	3.0 (Fig V-14)	3.0
Percentage overshoot in pitch angle (truth model=linear 4th-order actuators *)	90 † (Fig V-6)	100 † (Fig V-33)	105 ‡ (Fig V-34)
Percentage of initial condition still present in oscillation at 4 seconds (truth model=.6 mach/20,000 feet)	5 (Fig V-7)	None	None
Percentage of initial condition still present in oscillation at 4 seconds (truth model=.6 mach/30,000 feet)	15 (Fig V-8)	30	25

* Truth model is otherwise identical to linear design model.
† Oscillatory.
‡ Non-oscillatory.

Table V-3. Summary of Minimum Singular Values (Inverse Return Difference) and Regulator Pole Locations for Standard PI Regulator Designs

Design	$\underline{\sigma}_{.01}$	$\underline{\sigma}_{1.0}$	$\underline{\sigma}_{10}$	$\underline{\sigma}_{100}$	$\underline{\sigma}_{min}$	$\omega_{\sigma_{min}}$	Regulator Poles *
SR-B	0.99	0.91	12.6	225	0.895	0.8	-2.26 -55.0 -3.85 -73.6 -19.6 -29.8 ± j43.0
SR-2	0.99	1.0	11.2	280	0.86	0.57	-1.3 -37.7 -2.4 -100.4 -19.9 -33.8 ± j44.9
SR-3	0.99	0.92	17.8	225	0.86	0.67	-2.1 ± j0.03 -19.9 -30.3 -67.5 -27.5 ± j36.3

* Refer to Appendix E.

Table V-4. Definition of Designs Using Implicit Model-Following with a Two-State Regulator Command Model

Design	Quadratic Weights	Command Model	Gains (see Appendix E)	
			Regulator	CGT
IMF2-1	$\underline{Q}_I$=diag(25,25) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(.5,2)	P=5	$\underline{K}_x$: -43.79 30.5 -1.994 1.804 4.18E-2 53.98 -56.2 0.3102 2.4E-2 0.654	$\underline{K}_{xm}$: -1.537 -24.09 -12.18 38.66
			$\underline{K}_z$: 0.1514 -1.273 -1.751 2.434	$\underline{K}_{xu}$: -6.9 -11.26 -1.376 28.86
IMF2-2	$\underline{Q}_I$=diag(25,25) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(.5,2)	P=2	$\underline{K}_x$: -41.45 30.19 -1.949 1.793 4.0E-2 52.03 -54.24 0.3019 2.34E-2 0.6456	$\underline{K}_{xm}$: 0.2665 -23.79 -12.08 36.81
			$\underline{K}_z$: 0.7217 -1.187 -1.675 1.894	$\underline{K}_{xu}$: -6.683 -11.24 -1.412 28.7
IMF2-3	$\underline{Q}_I$=diag(10,10) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(1,2)	P=2	$\underline{K}_x$: -26.77 18.82 -1.296 1.16 3.53E-2 49.16 -51.28 0.2872 2.8E-2 0.6291	$\underline{K}_{xm}$: -0.1783 -14.11 -11.8 34.08
			$\underline{K}_z$: 0.3942 -0.7404 -1.696 1.773	$\underline{K}_{xu}$: -4.063 -8.424 -1.47 28.35
IMF2-4	$\underline{Q}_I$=diag(3,5) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(2,3)	P=2	$\underline{K}_x$: -16.56 11.25 -0.8446 0.7241 3.24E-2 36.52 -38.01 0.2076 2.86E-2 0.4708	$\underline{K}_{xm}$: -0.099 -7.652 -9.797 22.85
			$\underline{K}_z$: 0.1902 -0.4433 -1.272 1.312	$\underline{K}_{xu}$: -2.241 -6.565 -1.755 25.22
IMF2-5	$\underline{Q}_I$=diag(5,5) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(1,2)	P=1.5	$\underline{K}_x$: -24.06 18.8 -1.192 1.124 2.7E-2 48.28 -50.13 0.2959 2.18E-2 0.6212	$\underline{K}_{xm}$: 2.016 -14.08 -11.53 33.01
			$\underline{K}_z$: 0.5441 -0.7236 -1.697 1.712	$\underline{K}_{xu}$: -3.573 -8.425 -1.427 28.22

Table V-5. Summary of Performance Analysis for Implicit Designs with a Two-State Regulator Command Model

Condition and Measure of Performance	IMF2-1	IMF2-2	IMF2-3	IMF2-4	IMF2-5
Percentage overshoot in pitch angle (truth model=linear design model)	42	34	37	50	28
Rise time/peak time (seconds) for pitch angle (truth model=linear design model)	0.12/0.24	0.12/0.26	0.16/0.30	0.18/0.36	0.18/0.36
Pitch angle initial condition (degrees) to cause 100% overshoot (nonlinear actuators*)	0.4	0.5	0.6	0.7	0.6
Maximum pitch angle initial condition (degrees) for stability (nonlinear actuators*)	0.7	0.9	1.1	1.5	1.7
Maximum CGT pitch input (degrees) for stability (nonlinear actuators*)	3 (Fig V-17)	3	3 (Fig V-21)	3 (Fig V-23)	3
Percentage overshoot in pitch angle (truth model=linear 4th-order actuators*)	80 † (Fig V-35)	70 †	60 ‡ (Fig V-36)	70 ‡ (Fig V-37)	35 ‡
Percentage of initial condition still present in oscillation at 4 seconds (truth model=.6 mach/20,000 feet)	15	None	None	10 (Fig V-42)	10
Percentage of initial condition still present in oscillation at 4 seconds (truth model=.6 mach/30,000 feet)	500 Di-verging	140 Con-verging	300 Di-verging	550 Di-verging	800 Di-verging

* Truth model is otherwise identical to linear design model.

† oscillatory.

‡ non-oscillatory.

Table V-6. Summary of Minimum Singular Values (Inverse Return Difference) and Regulator Pole Locations for Implicit Designs with a Two-State Regulator Command Model

Design	$\underline{\sigma}_{0.1}$	$\underline{\sigma}_{1.0}$	$\underline{\sigma}_{10}$	$\underline{\sigma}_{100}$	$\underline{\sigma}_{min}$	$\omega_{\underline{\sigma}_{min}}$	Regulator Poles **
IMF2-1	0.98	0.7	15.8*	160	0.69	0.85	-4.2 -93.9 -5.0 -31.7 ± j12.6 -32.8 ± j37.2
IMF2-2	0.98	0.73	14.1*	160	0.69	0.77	-2.0 -93.9 -2.2 -31.8 ± j12.6 -32.9 ± j37.0
IMF2-3	0.99	0.85	17.8*	225*	0.8	0.75	-2.0 -61.4 -2.3 -25.7 ± j28.8 -30.1 ± j5.7
IMF2-4	0.99	0.88	28.2*	400*	0.79	0.73	-2.0 -42.5 -2.6 -19.8 ± j20.2 -25.6 ± j4.5
IMF2-5	0.99	1.05*	24.0*	250*	0.79	0.65	-1.5 -33.5 -2.3 -58.8 -23.2 ± j23.6 -25.2

* Singular values at this frequency equals or exceeds the value for the baseline (SR-B) design.

** Refer to Appendix E.

Table V-7. Definition of Designs Using Implicit Model-Following with a Three-State Regulator Command Model

Design	Quadratic Weights	Command Model	Gains (see Appendix E)	
			Regulator	CGT
IMF3-1	$\underline{Q}_I$=diag(10,10,.1) $\underline{R}_I$=diag(1,1,1) $\underline{U}_R$=diag(1,2,1)	P_1=5 P_2=5 P_3=10	$\underline{K}_x$: $\begin{matrix} -4.15E-2 & 18.46 & -2.463 & 1.422 & 0.1039 \\ 3.62E-2 & -52.59 & 0.6045 & 0.1342 & 0.6610 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} 13.89 & -13.91 & 0 \\ -35.26 & 35.27 & 0 \end{matrix}$
			$\underline{K}_z$: $\begin{matrix} -0.2512 & -0.815 & -0.7184 \\ -1.791 & 2.063 & 0.9735 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} 8.259 & -8.273 & 0 \\ -28.47 & 28.49 & 0 \end{matrix}$
IMF3-2	$\underline{Q}_I$=diag(10,20,.01) $\underline{R}_I$=diag(1,1,1) $\underline{U}_R$=diag(1,2,1)	P_1=10 P_2=10 P_3=15	$\underline{K}_x$: $\begin{matrix} -3.55E-2 & 19.7 & -1.895 & 1.217 & 4.54E-2 \\ 4.28E-2 & -58.1 & 0.8622 & 8.65E-2 & 0.6739 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} 14.94 & -14.96 & 0 \\ -40.47 & 40.48 & 0 \end{matrix}$
			$\underline{K}_z$: $\begin{matrix} -0.8035 & -0.9805 & -0.6824 \\ -1.808 & 3.206 & 1.122 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} 8.476 & -8.488 & 0 \\ -28.94 & 28.96 & 0 \end{matrix}$
IMF3-3	$\underline{Q}_I$=diag(5,20,.01) $\underline{R}_I$=diag(1,1,1) $\underline{U}_R$=diag(1,2,1)	P_1=10 P_2=10 P_3=15 $\underline{A}(1,3)$ =1	$\underline{K}_x$: $\begin{matrix} -3.125E-2 & 19.81 & -1.672 & 1.167 & 3.2E-2 \\ 4.38E-2 & -58.07 & 0.9149 & 7.47E-2 & 0.6707 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} 15.02 & -15.04 & 0 \\ -40.45 & 40.46 & 0 \end{matrix}$
			$\underline{K}_z$: $\begin{matrix} -0.3161 & -0.9922 & -0.5784 \\ -1.694 & 3.203 & 1.147 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} 8.512 & -8.524 & 0 \\ -28.93 & 28.96 & 0 \end{matrix}$
IMF3-4	$\underline{Q}_I$=diag(5,10,.01) $\underline{R}_I$=diag(1,1,1) $\underline{U}_R$=diag(5,5,5)	P_1=5 P_2=5 P_3=10	$\underline{K}_x$: $\begin{matrix} -2.23E-2 & 7.832 & -1.302 & 0.7212 & 7.67E-2 \\ 2.23E-2 & -31.76 & 0.386 & 9.95E-2 & 0.3883 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} 4.853 & -4.866 & 0 \\ -17.86 & 17.86 & 0 \end{matrix}$
			$\underline{K}_z$: $\begin{matrix} -0.6044 & -0.4836 & -0.4108 \\ -0.967 & 1.431 & 0.5988 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} 5.584 & -5.592 & 0 \\ -23.25 & 23.26 & 0 \end{matrix}$
IMF3-5	$\underline{Q}_I$=diag(1,5,.01) $\underline{R}_I$=diag(.05,.01, .05) $\underline{U}_R$=diag(5,5,5)	P_1=5 P_2=5 P_3=10	$\underline{K}_x$: $\begin{matrix} -1.12E-2 & 2.411 & -0.7375 & 0.4079 & 6.65E-2 \\ 4.93E-3 & -7.652 & 3.84E-2 & 6.78E-2 & 8.93E-2 \end{matrix}$	$\underline{K}_{xm}$: $\begin{matrix} 0.1738 & -0.1783 & 0 \\ 2.471 & -2.481 & 0 \end{matrix}$
			$\underline{K}_z$: $\begin{matrix} -0.2953 & -0.1664 & -0.1851 \\ -8.81E-2 & 0.5361 & 0.1481 \end{matrix}$	$\underline{K}_{xu}$: $\begin{matrix} 4.349 & -4.354 & 0 \\ -17.49 & 17.5 & 0 \end{matrix}$

Table V-8. Summary of Performance Analysis for Implicit Designs with a Three-State Regulator Command Model

Condition and Measure of Performance	IMF3-1	IMF3-2	IMF3-3	IMF3-4	IMF3-5
Percentage overshoot in pitch angle (truth model=linear design model)	2	5	15	5	0
Rise time/peak time (seconds) for pitch angle (truth model=linear design model)	0.62/1.34	0.32/0.44	0.36/0.54	0.34/0.82 (Fig V-29)	0.88/-- (Fig V-30)
Maximum pitch angle initial condition (degrees) for stability (nonlinear actuators *)	6.5	2.5	4.0	3.5	6.5
Maximum CGT pitch input (degrees) for stability (nonlinear actuators *)	1.5	1.0	1.5	1.5	2.5
Percentage overshoot in pitch angle (truth model=linear 4th-order actuators *)	Unstable (Fig V-38)	None ‡ (Fig V-39)	10 ‡	20 † (Fig V-40)	10 † (Fig V-41)
Percentage of initial condition still present in oscillation at 4 seconds (truth model=.6 mach/20,000 feet)	None (Fig V-43)	140 Di- verging	210 Di- verging	30	50
Percentage of initial condition still present in oscillation at 4 seconds (truth model=.6 mach/30,000 feet)	2 (Fig V-44)	1700 Di- verging	2500 Di- verging	320 Di- verging	340 Di- verging

* Truth model is otherwise identical to linear design model.
† Oscillatory.
‡ Non-oscillatory.

Table V-9. Summary of Minimum Singular Values (Inverse Return Difference) and Regulator Pole Locations for Implicit Designs with a Three-State Regulator Command Model

Design	$\underline{\sigma}_{0.1}$	$\underline{\sigma}_{1.0}$	$\underline{\sigma}_{10}$	$\underline{\sigma}_{100}$	$\underline{\sigma}_{min}$	$\omega_{\underline{\sigma}_{min}}$	Regulator Poles**
IMF3-1	0.99	0.86	12.6*	200	0.86	0.83	-3.6 ±j0.45 -13.6 -30.3 -54.9 -68.9 ±j67.4
IMF3-2	0.99	0.67	20.0*	200	0.67	0.85	-7.9 -40.6 -9.4 -54.9 -29.3 ±j9.96 -41.4 ±j30.5
IMF3-3	0.99	0.50	25.0*	200	0.50	0.9	-7.9 -54.9 -12.5 ±j7.5 -31.6 ±j8.9 -47.1 ±j28.9
IMF3-4	0.98	0.79	17.8*	315*	0.78	0.87	-4.0 -4.5 -22.7 -24.2 -22.4 ±j7.6 -26.1 ±j23.2
IMF3-5	0.89	0.55	31.6*	600*	0.51	0.6	-3.6 -5.0 -5.3 -13.8 -14.6 ±j4.5 -25.1 ±j23.3

* Singular value at this frequency equals or exceeds the value for the baseline (SR-B) design.

** Refer to Appendix E.

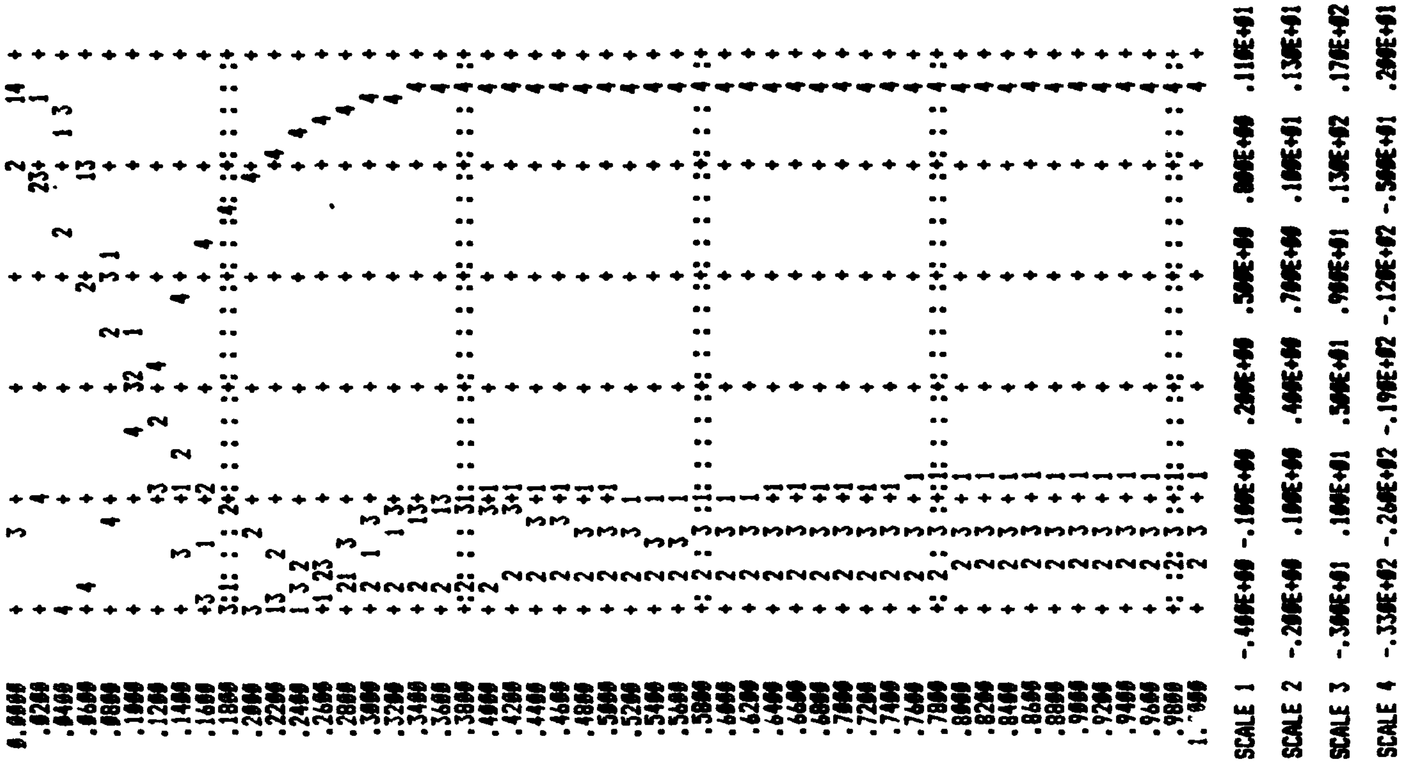


Figure V-1. Design SR-B, Linear Design Model Actuators, PI Regulator with Initial Condition = 1 Degree

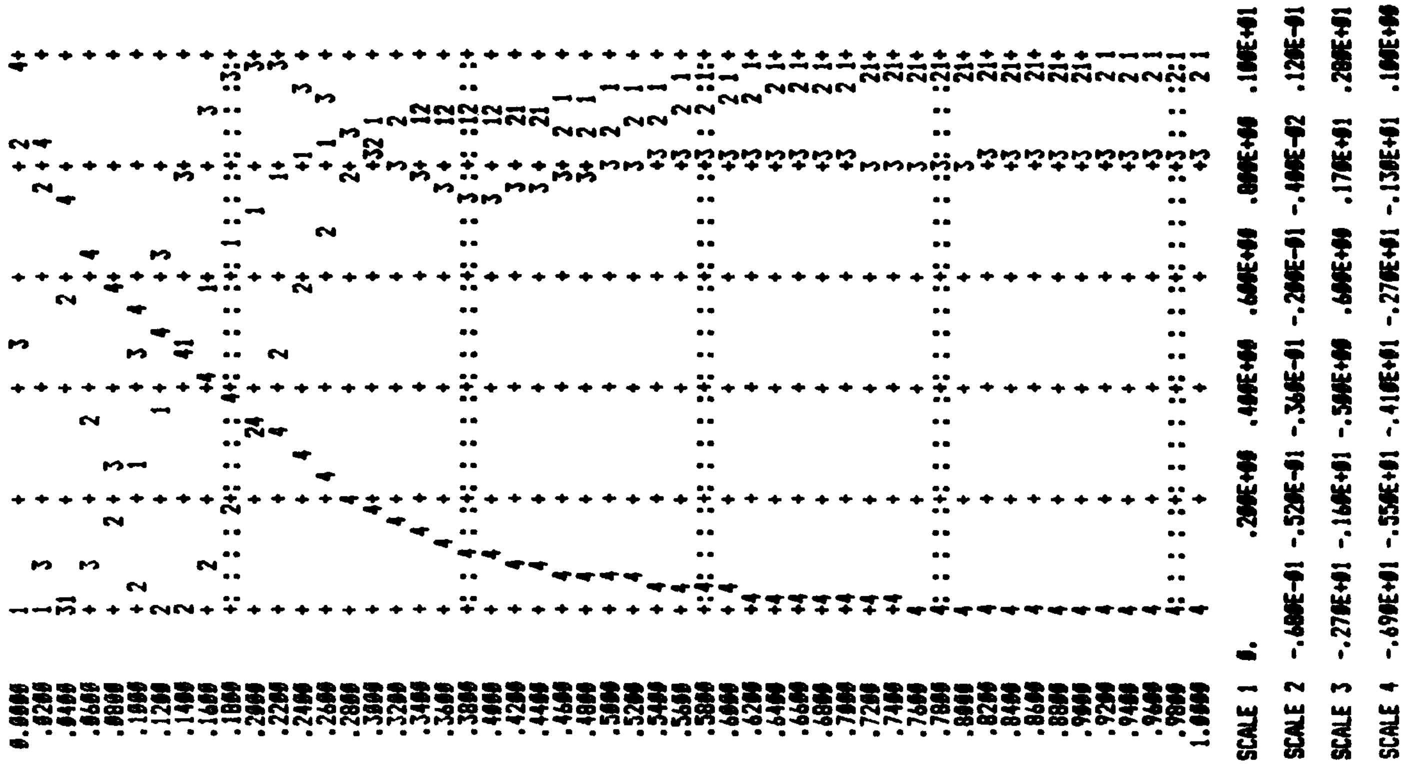


Figure V-2. Design SR-B, Linear Design Model Actuators,
CGT Response to Step Input = 1 Degree

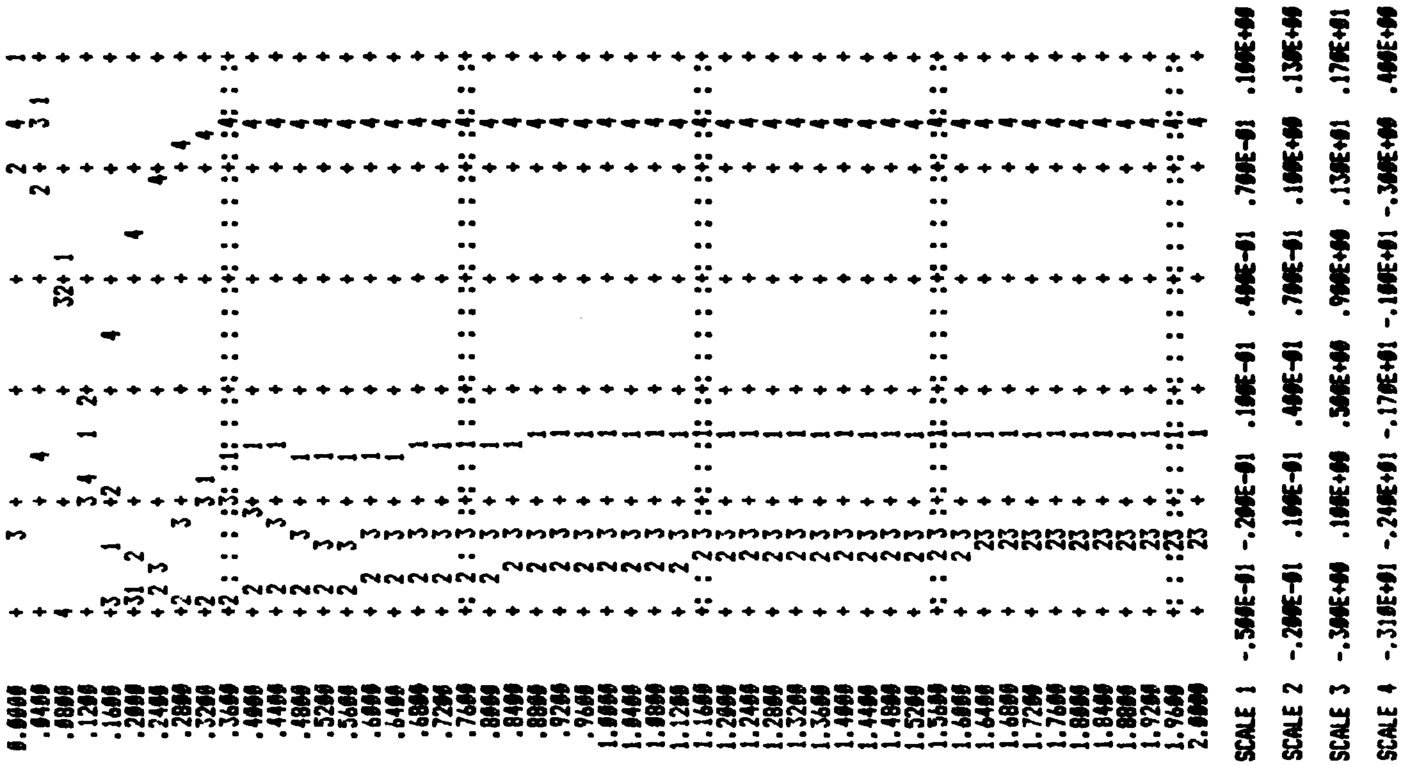


Figure V-3. Design SR-B, Nonlinear Single-State Actuators, Initial Condition = 0.1 Degree

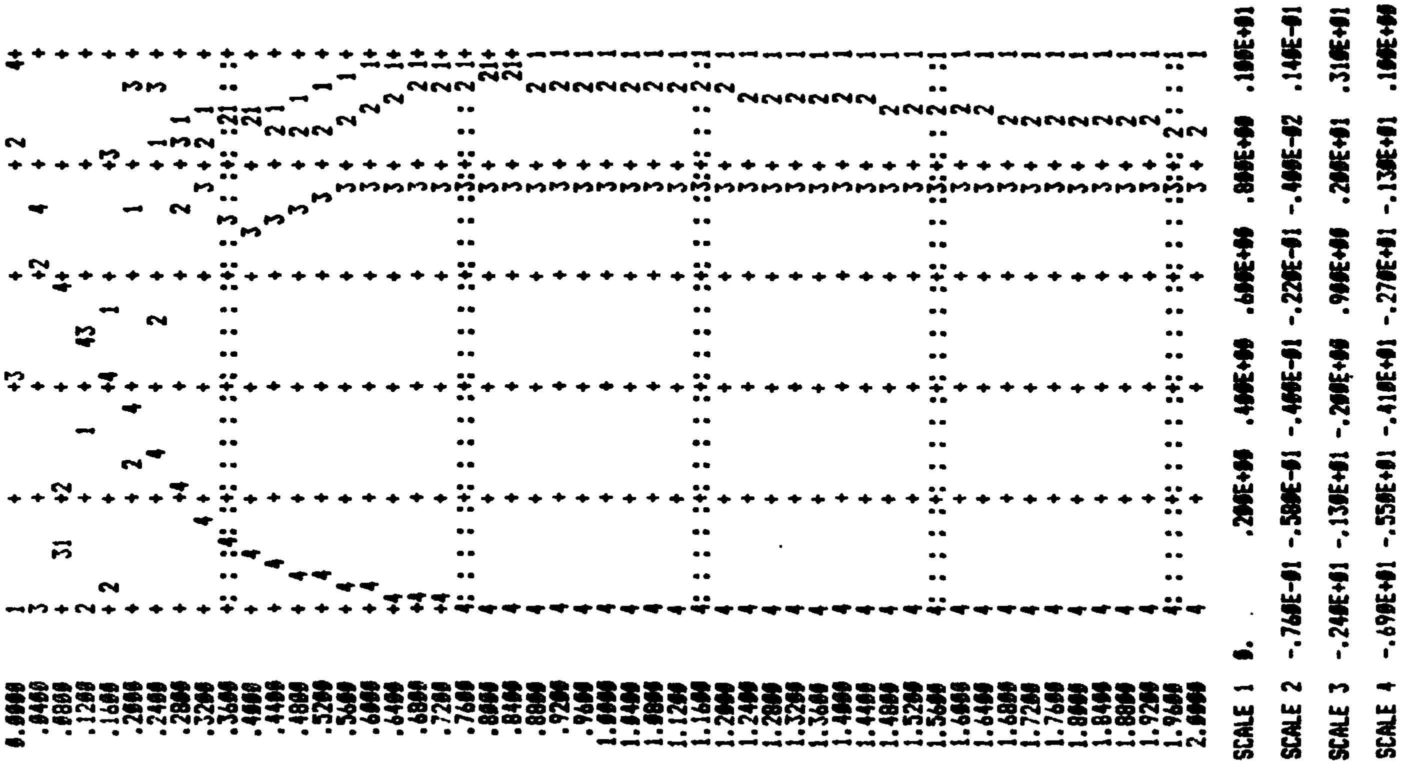


Figure V-4. Design SR-B, Nonlinear Single-State Actuators, CGT Step Input = 1 Degree

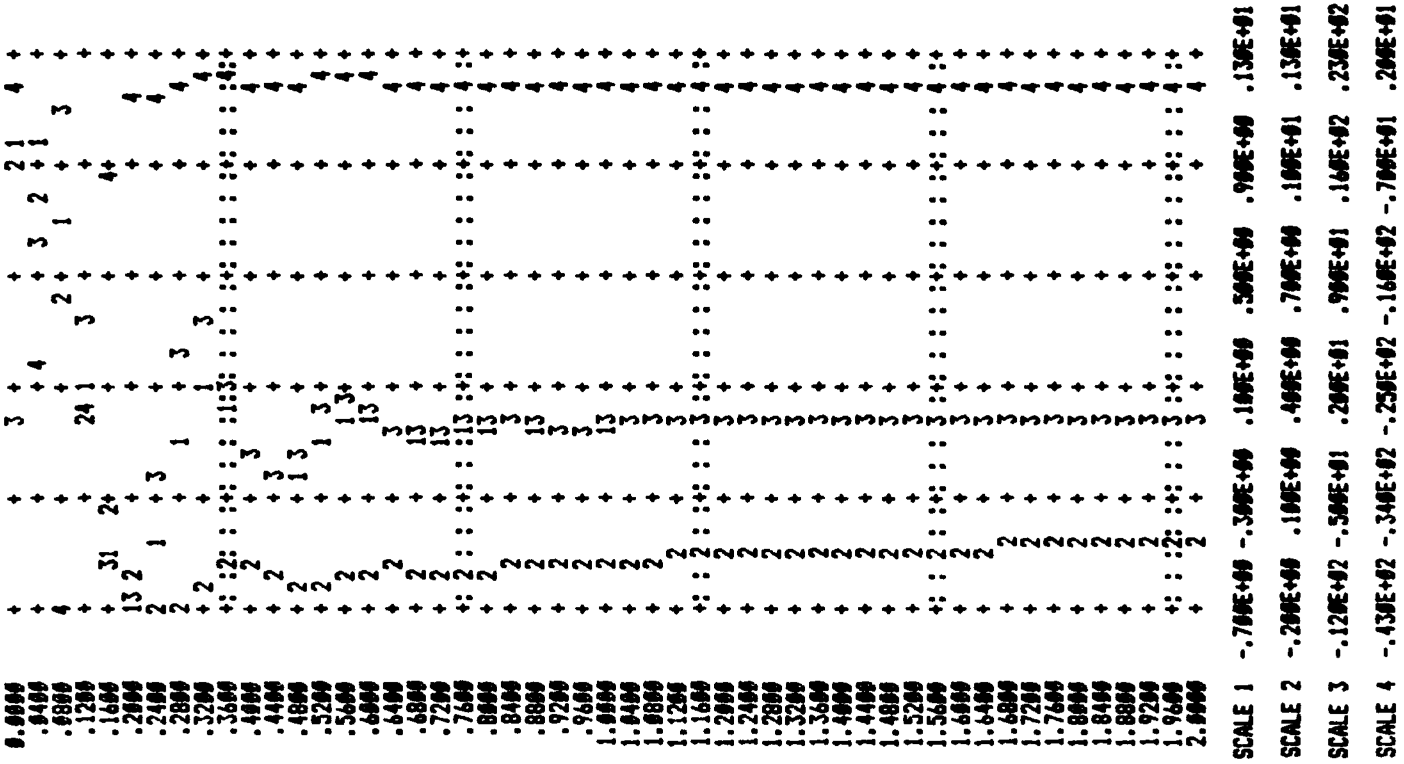


Figure V-5. Design SR-B, Linear Three-State Actuators, Initial Condition = 1 Degree

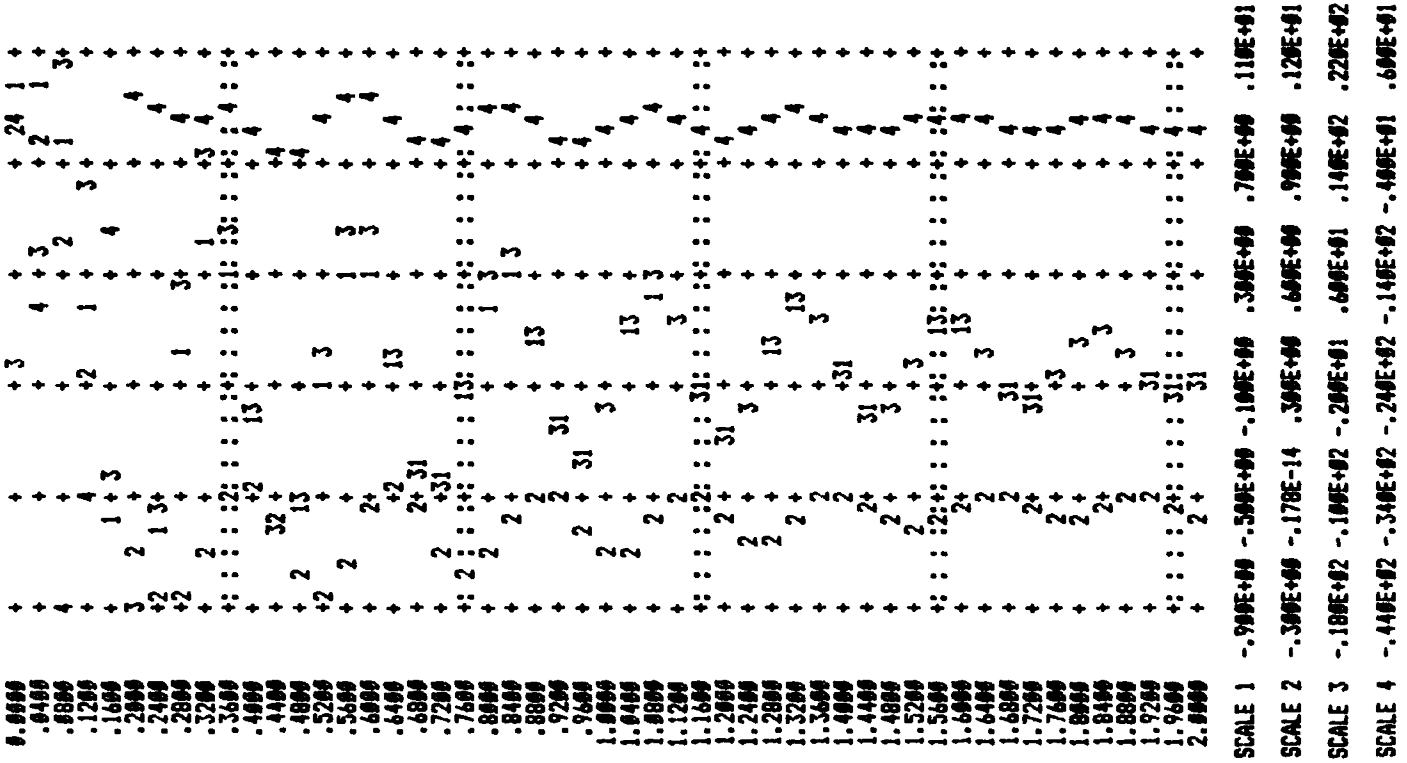


Figure V-6. Design SR-B, Linear Four-State Actuators,
Initial Condition = 1 Degree

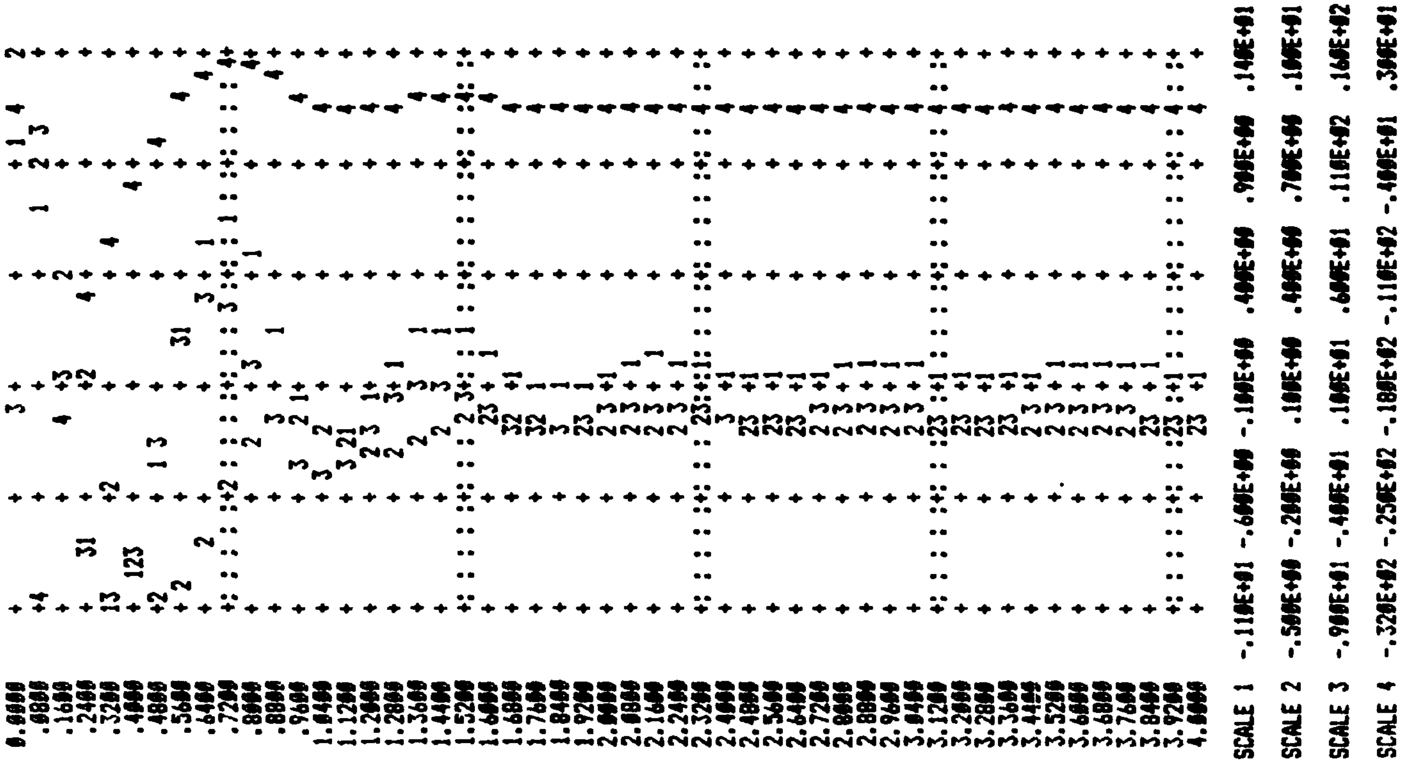


Figure V-7. Design SR-B, Linear Single-State Actuators,
Flight Condition = .6 Mach/20,000 feet,
Initial Condition = 1 Degree

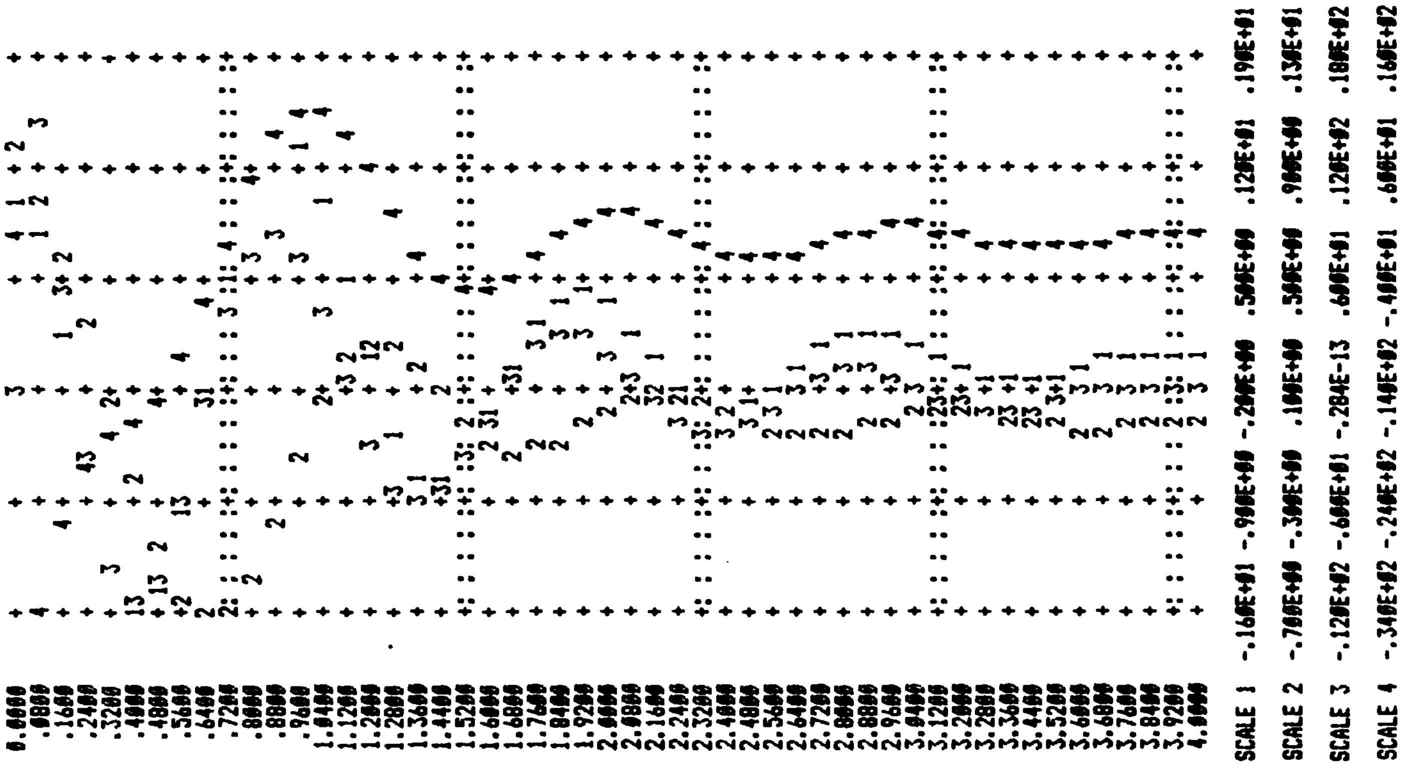


Figure V-8. Design SR-B, Linear Single-State Actuators, Flight Condition = .6 Mach/30,000 feet, Initial Condition = 1 Degree

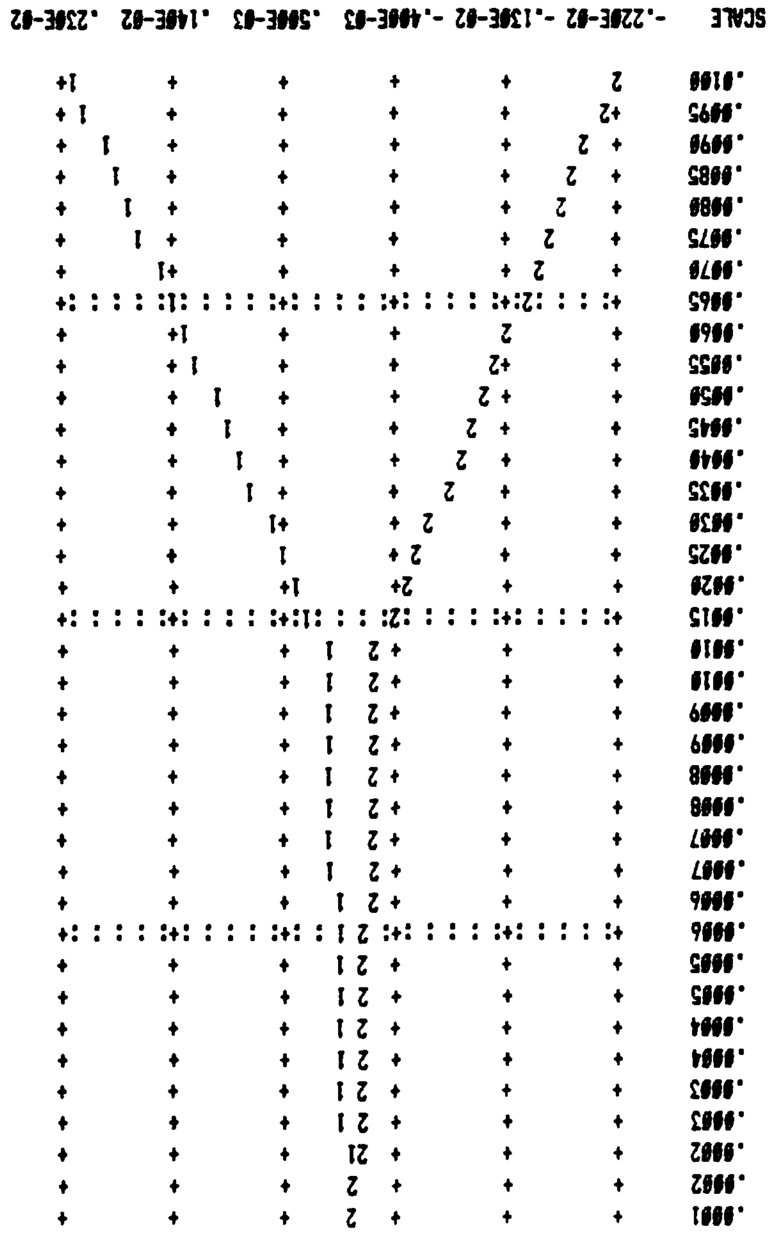


Figure V-9. Design SR-B, Singular Values of Inverse Return Difference Function Over Frequency Range of ω = .0001 to .01

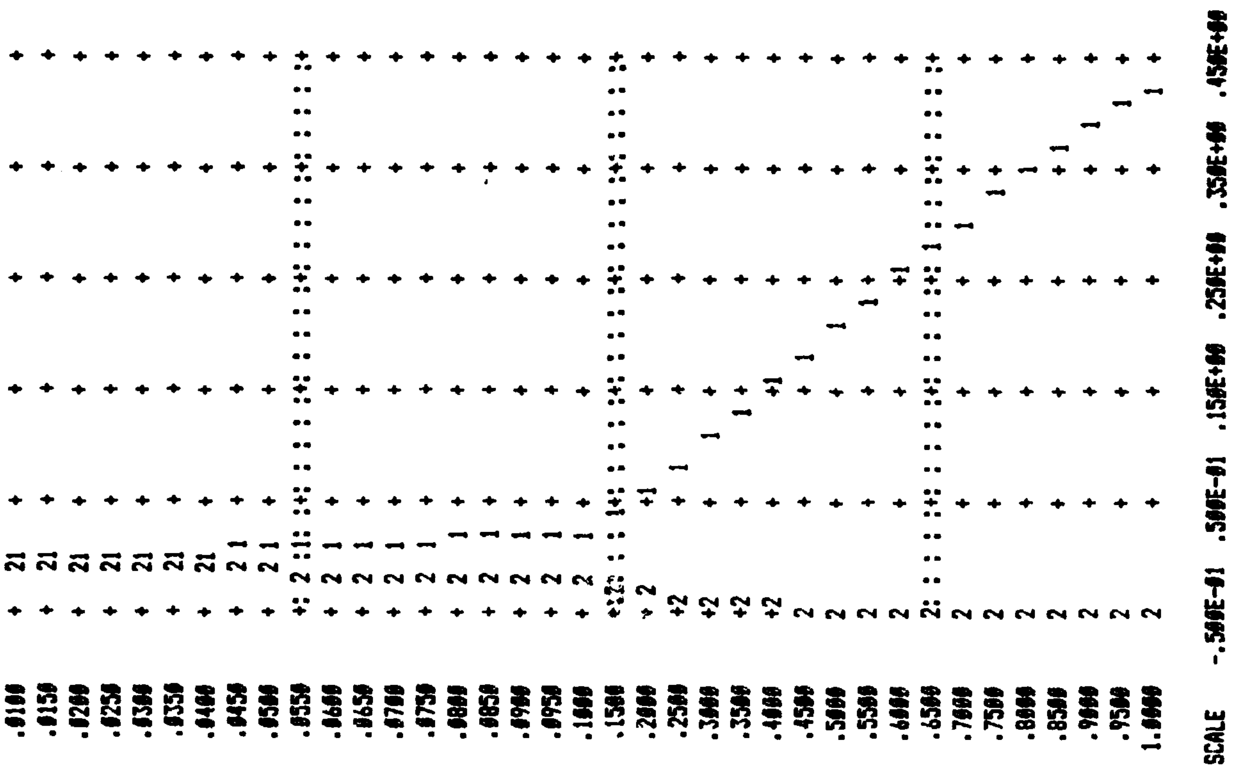


Figure V-10. Design SR-B, Singular Values ~~of~~ of Inverse Return Difference Function Over Frequency Range of ω = .01 to 1.0

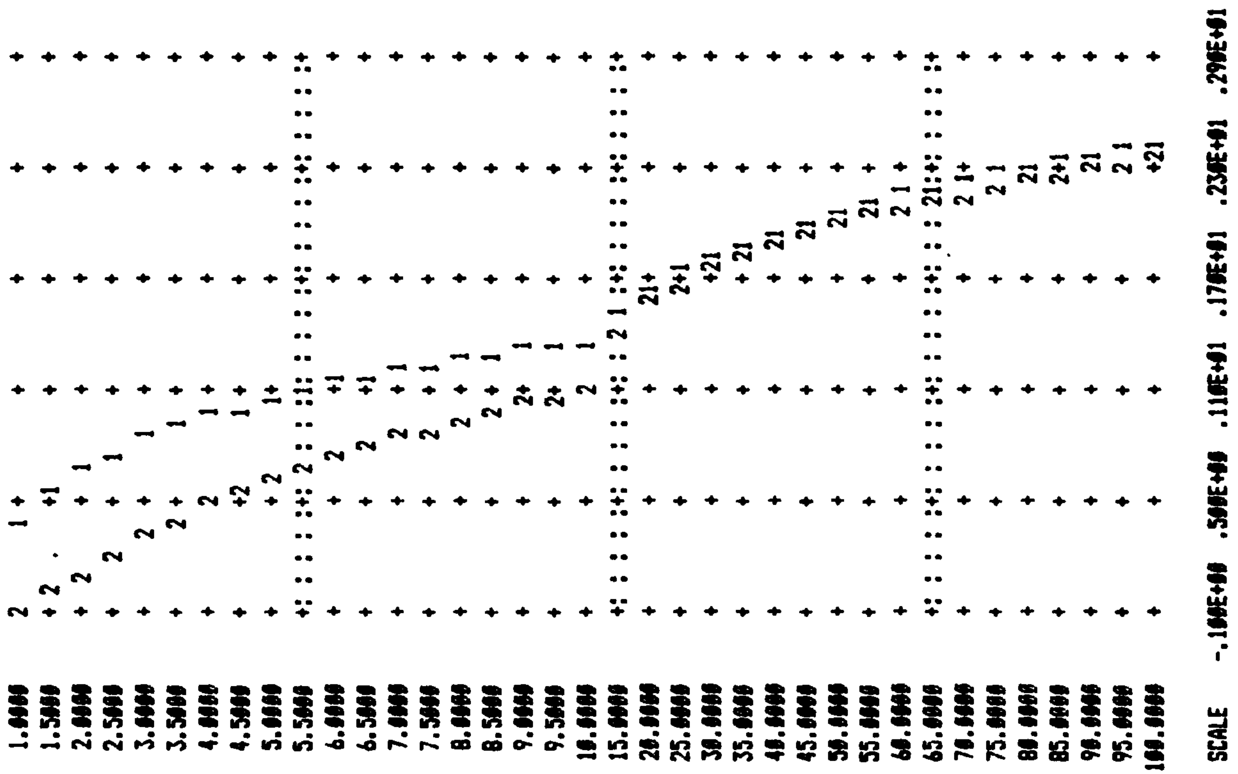


Figure V-11. Design SR-B, Singular Values of Inverse Return Difference Function Over Frequency Range of ω = 1.0 to 100.0

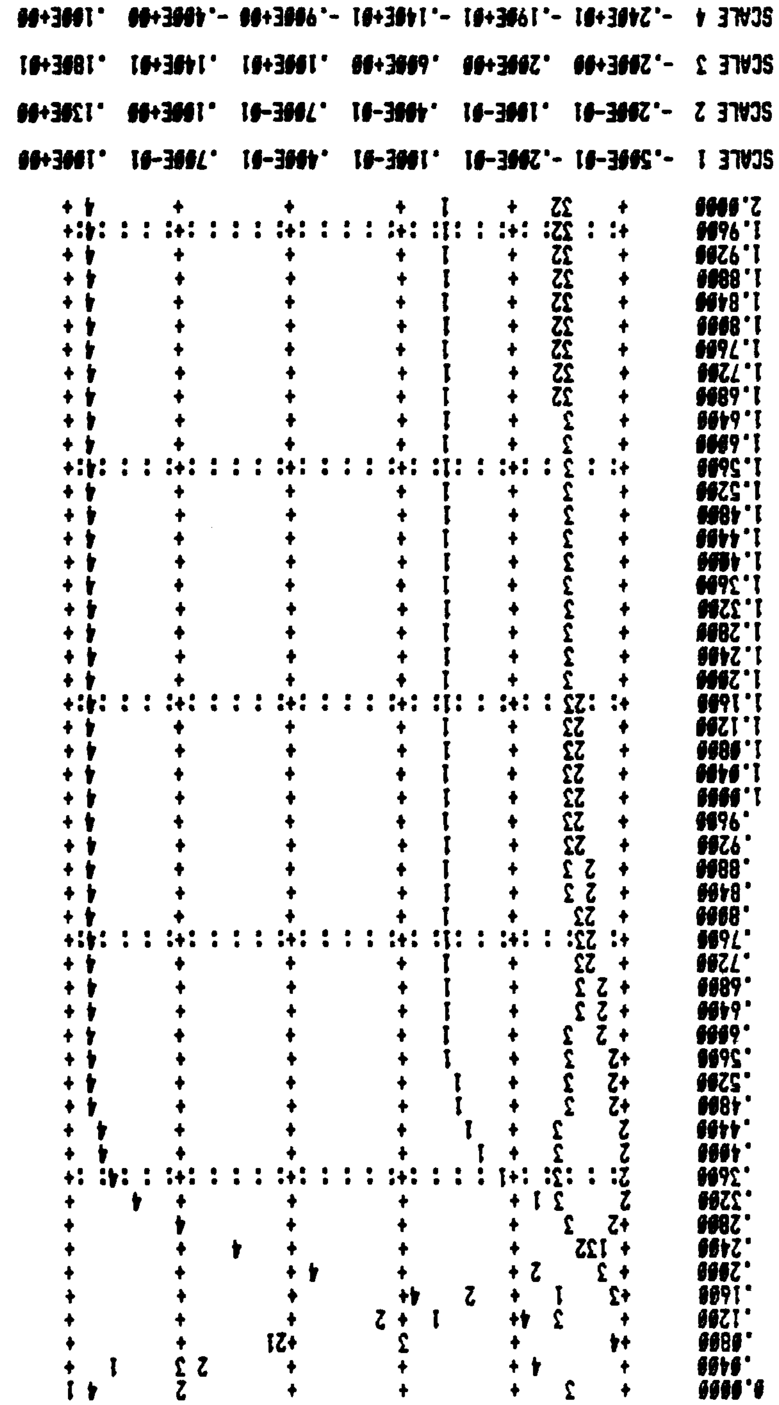


Figure V-12. Design SR-2, Nonlinear Single-State Actuators, Initial Condition = 0.1 Degree

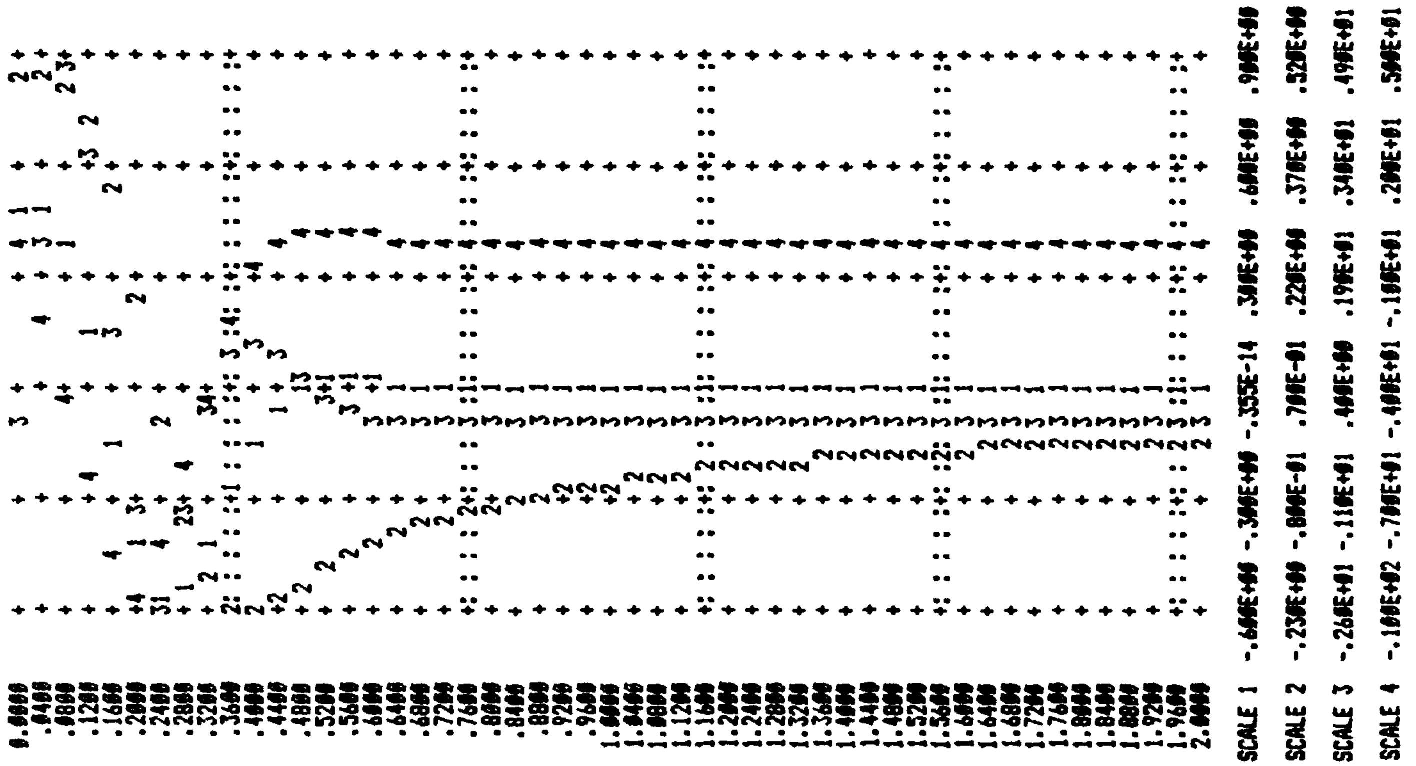


Figure V-13. Design SR-2, Nonlinear Single-State Actuators, Initial Condition = 0.5 Degree

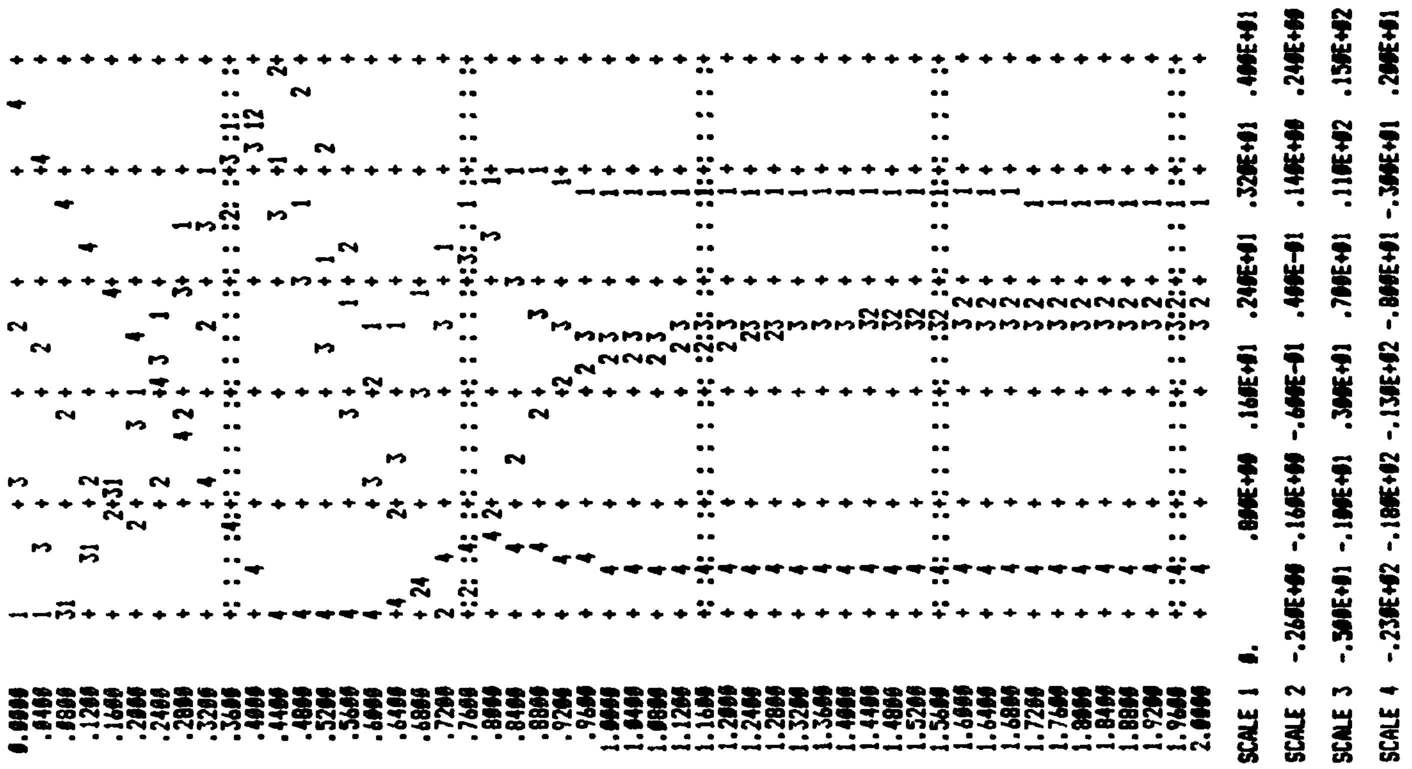


Figure V-14. Design SR-2, Nonlinear Single-State Actuators,
CGT Step Input = 3 Degrees

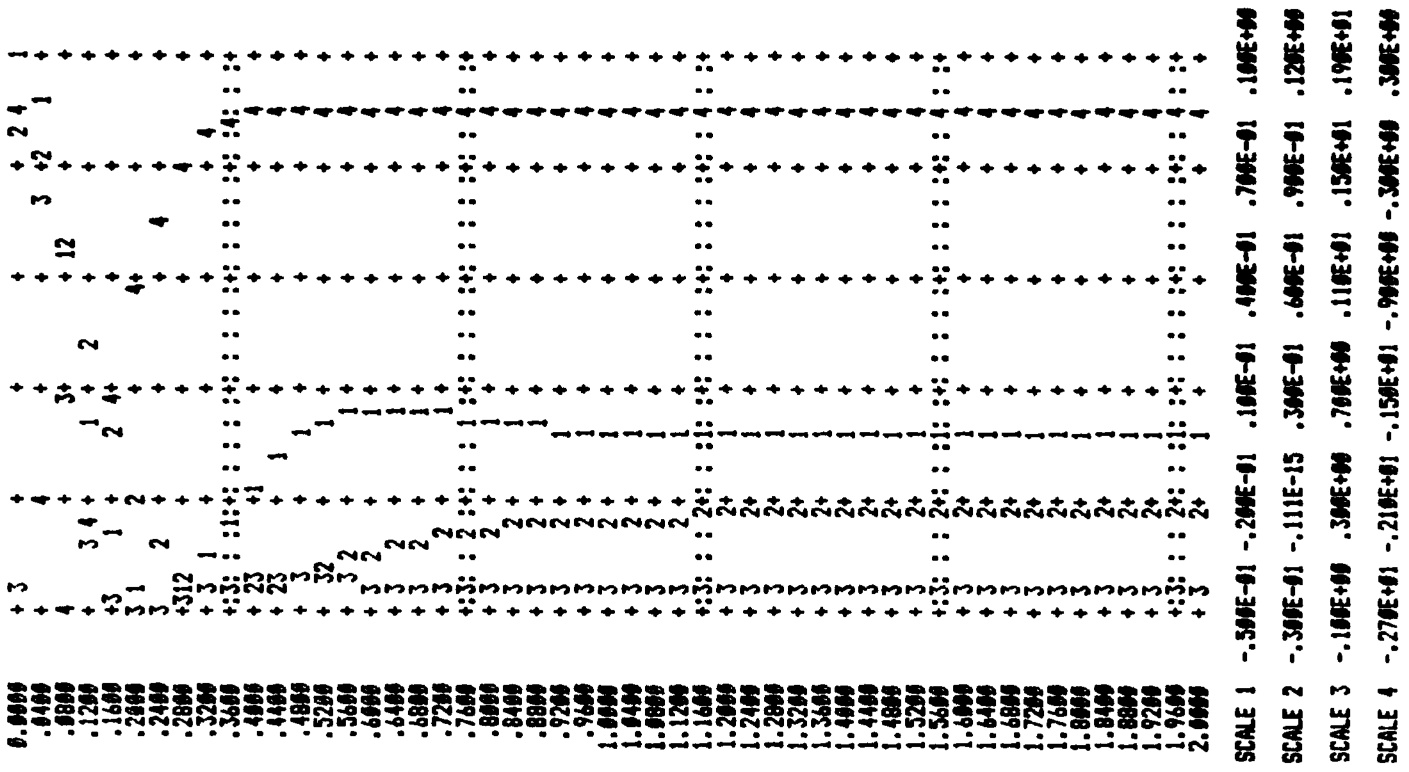


Figure V-15. Design IMF2-1, Nonlinear Single-State Actuators, Initial Condition = 0.1 Degree

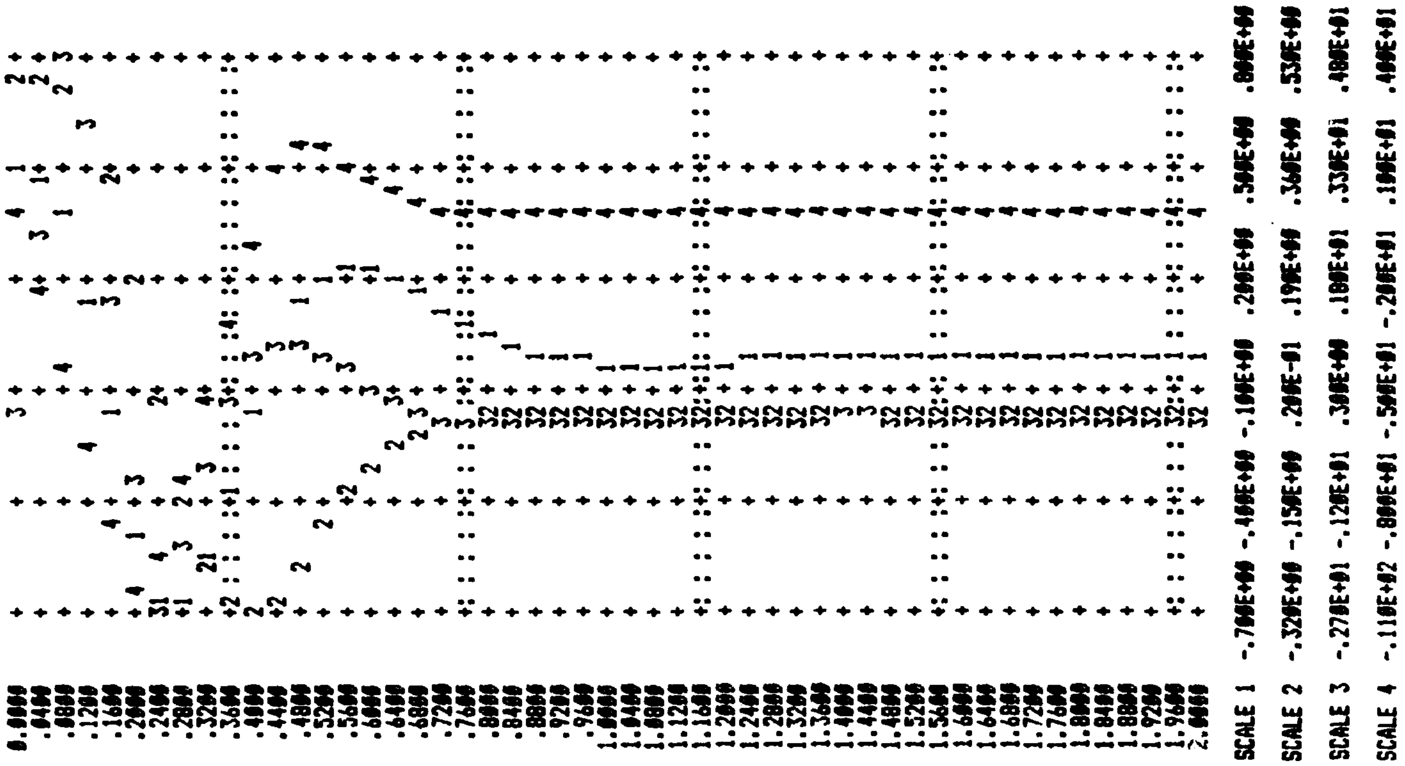


Figure V-16. Design IMF2-1, Nonlinear Single-State Actuators, Initial Condition = 0.5 Degree

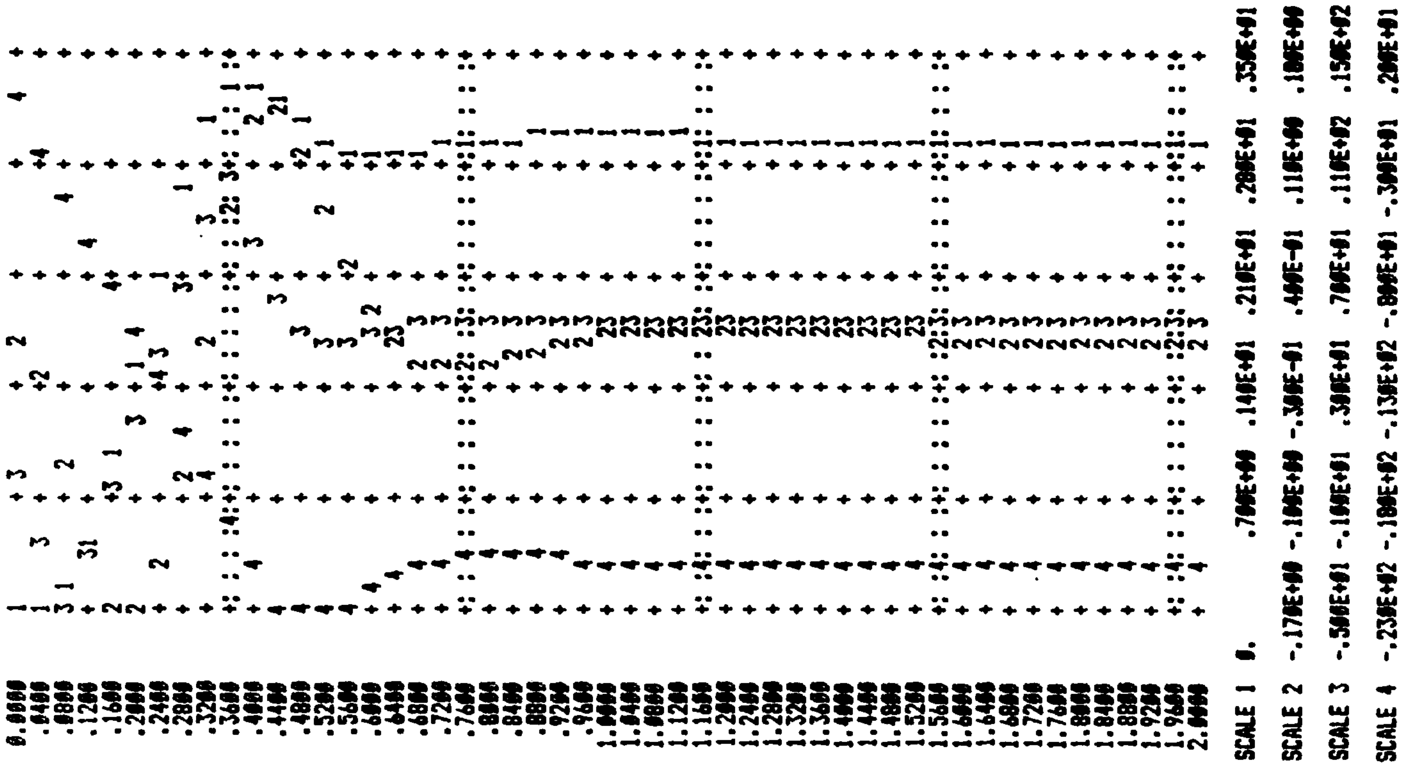


Figure V-17. Design IMF2-1, Nonlinear Single-State Actuators, CGT Step Input = 3 Degrees

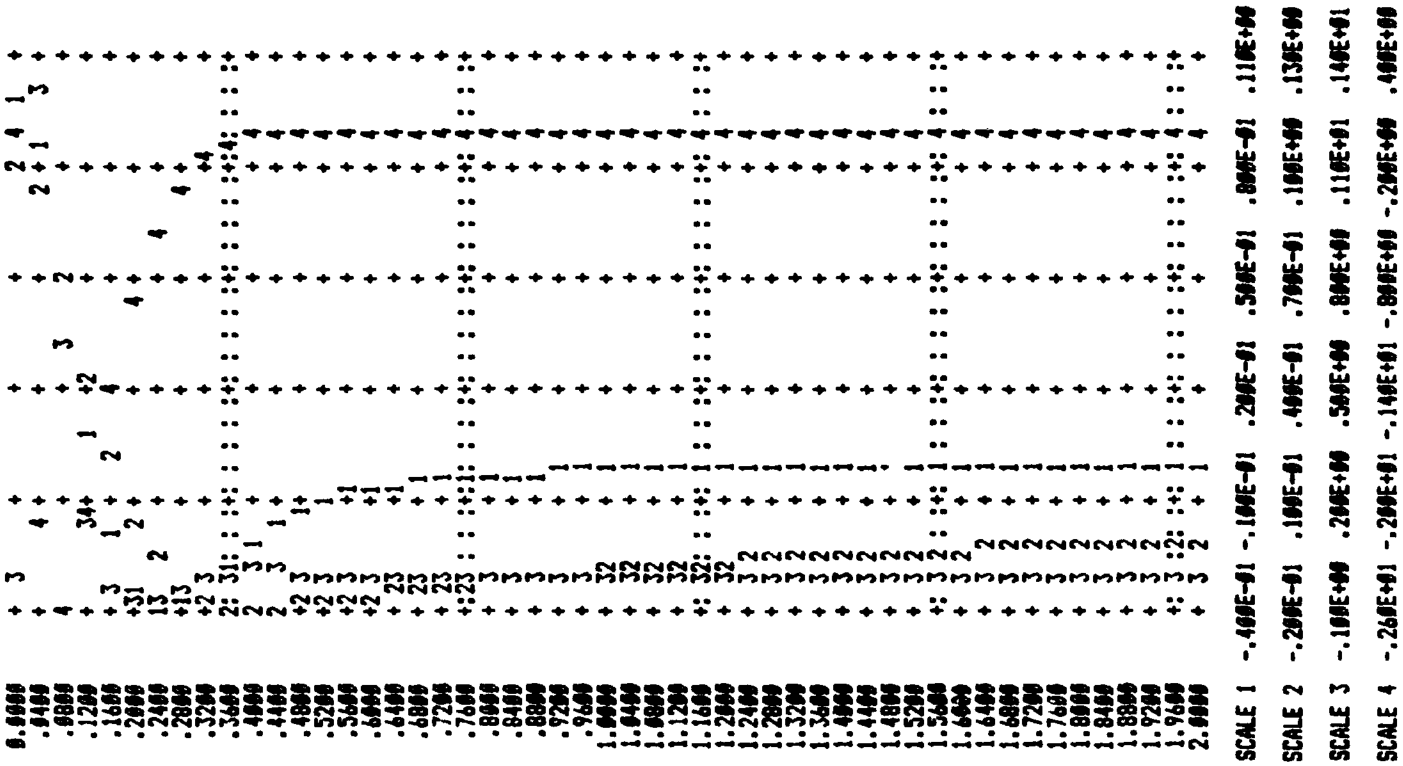


Figure V-18. Design IMF2-2, Nonlinear Single-State Actuators, Initial Condition = 0.1 Degree

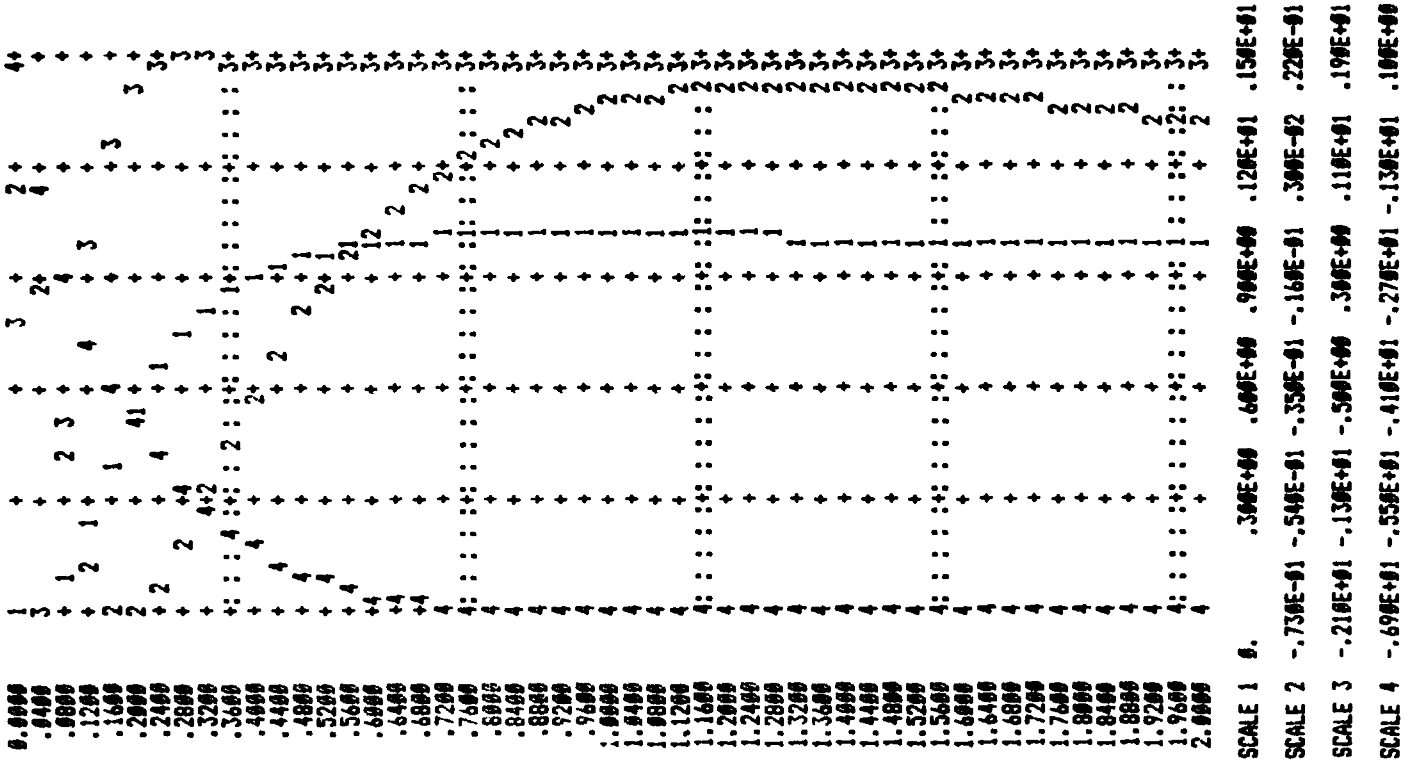


Figure V-19. Design IMF2-2, Nonlinear Single-State Actuators, CGT Step Input = 1 Degree

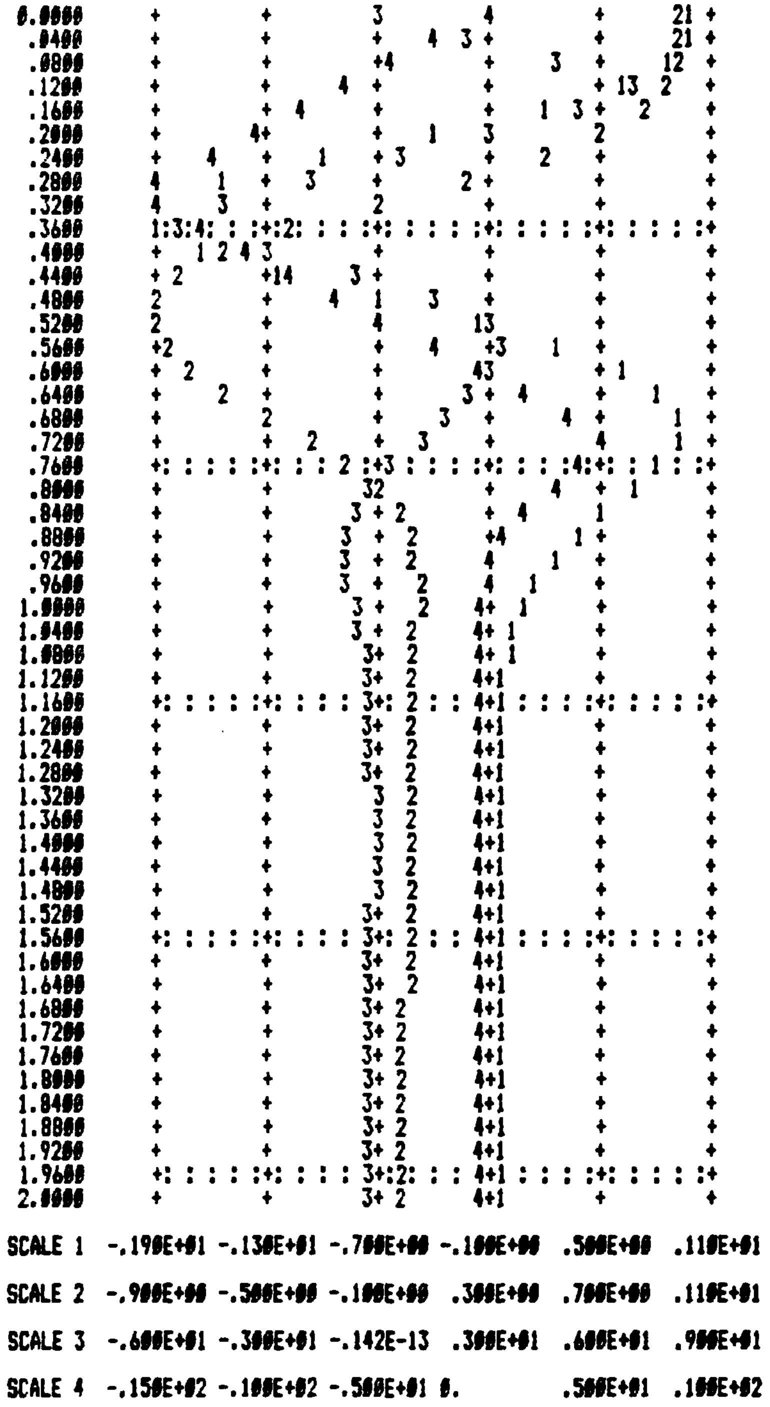


Figure V-20. Design IMF2-3, Nonlinear Single-State Actuators, Initial Condition = 1 Degree

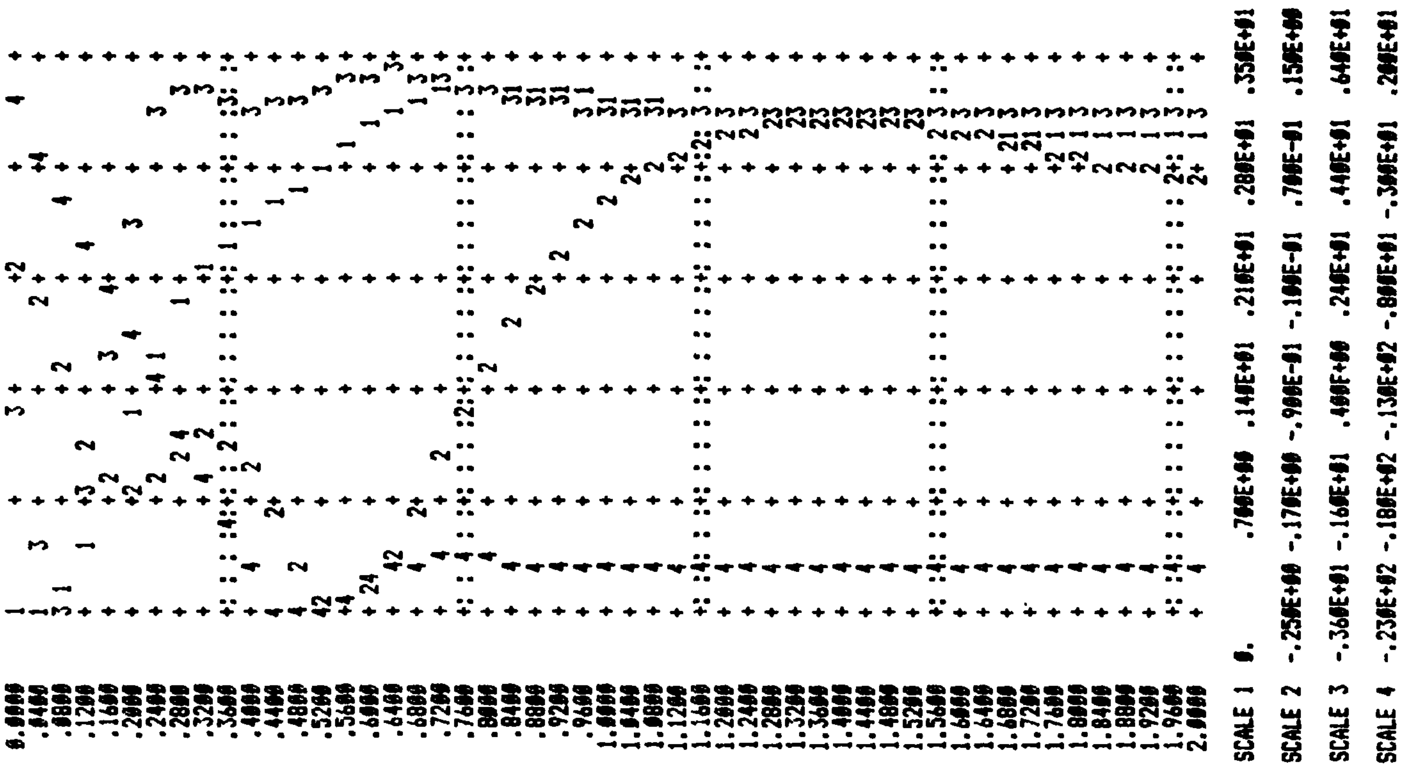


Figure V-21. Design IMF2-3, Nonlinear Single-State Actuators, CGT Step Input = 3 Degrees

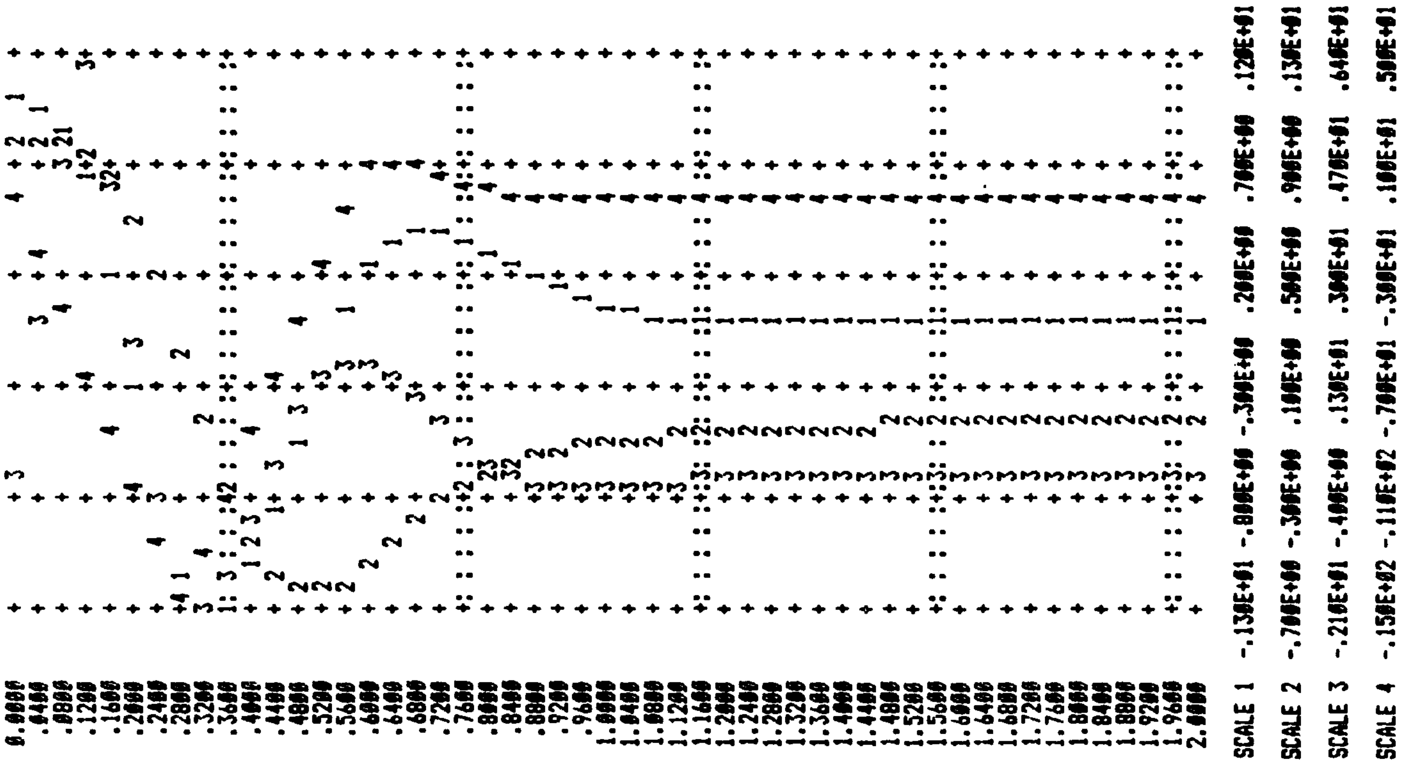


Figure V-22. Design IMF2-4, Nonlinear Single-State Actuators, Initial Condition = 1 Degree

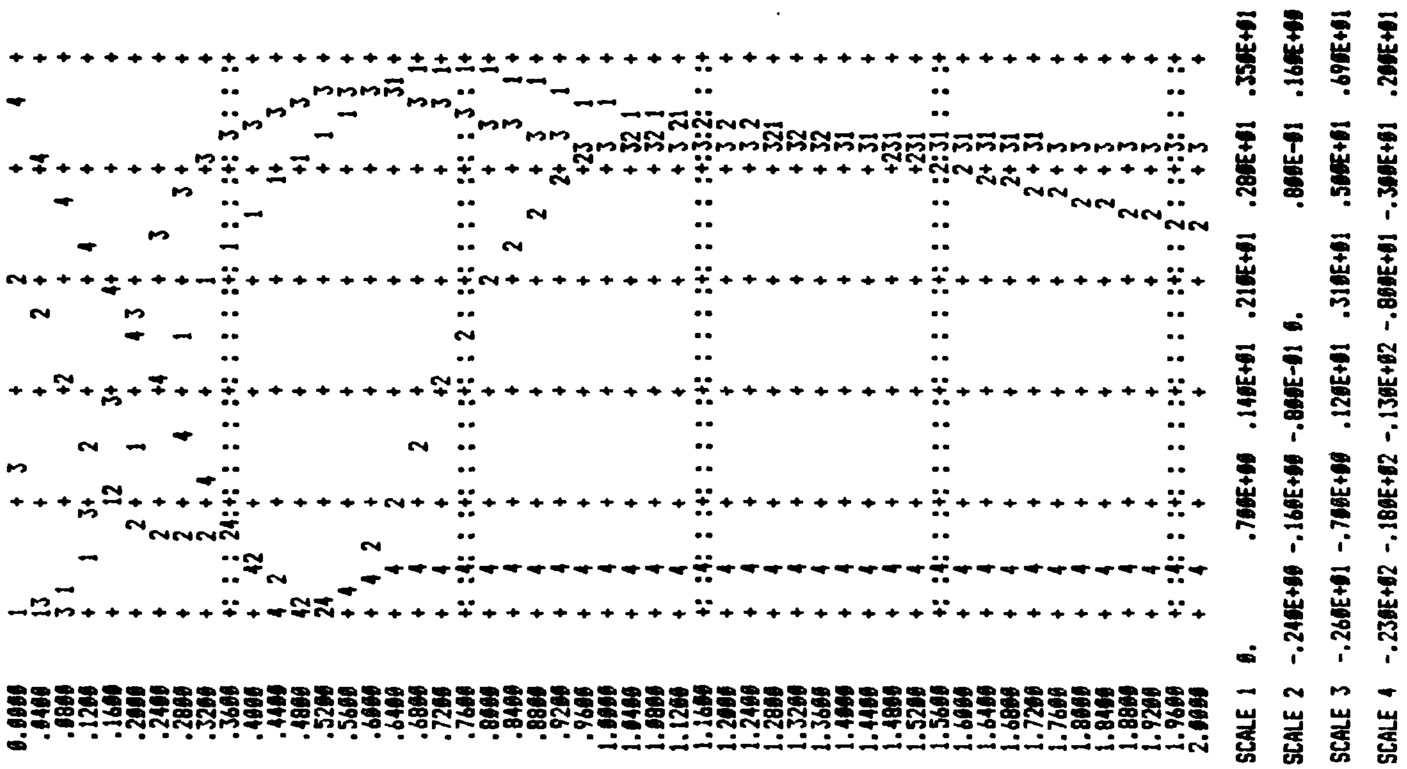


Figure V-23. Design IMF2-4, Nonlinear Single-State Actuators, CGT Step Input = 3 Degrees

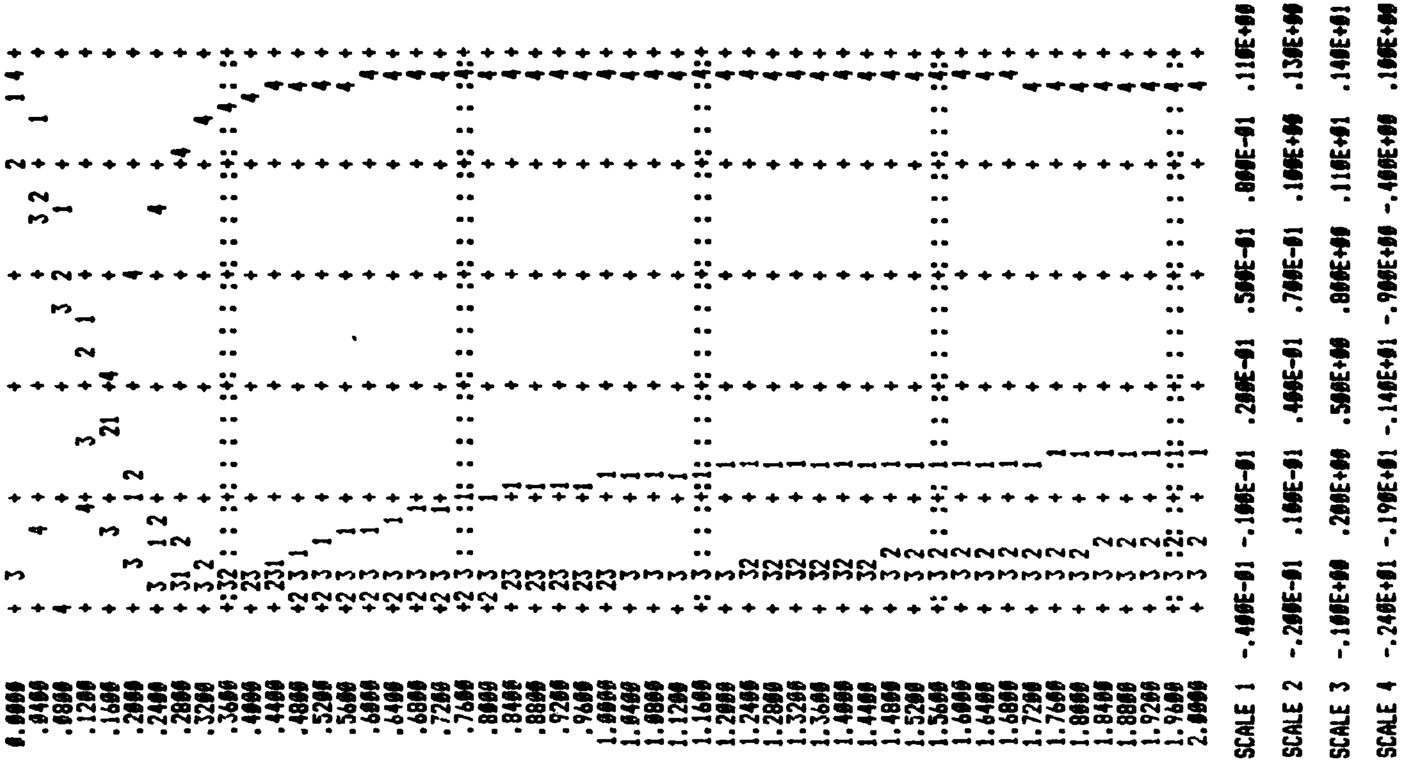


Figure V-24. Design IMF2-5, Nonlinear Single-State Actuators, Initial Condition = 0.1 Degree

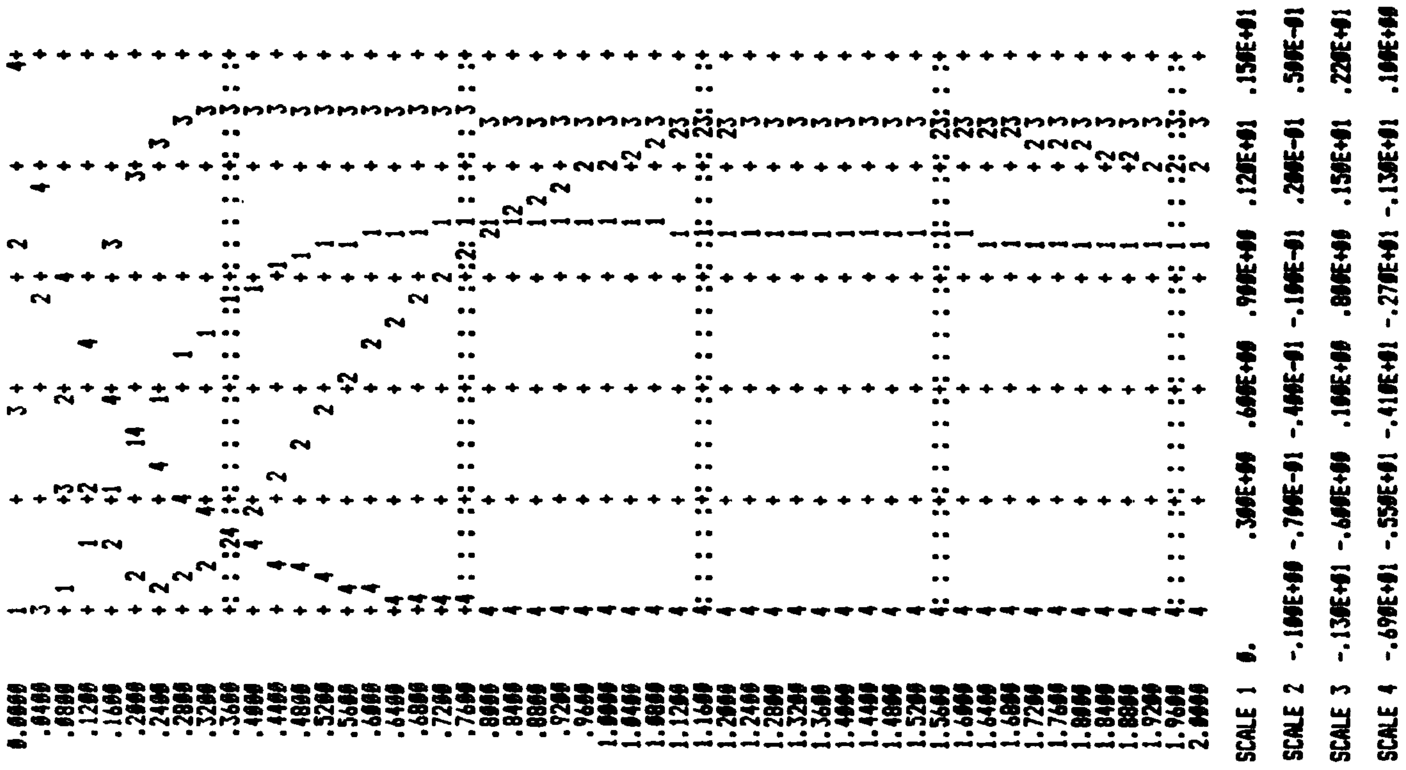


Figure V-25. Design IMF2-5, Nonlinear Single-State Actuators, CGT Step Input = 1 Degree

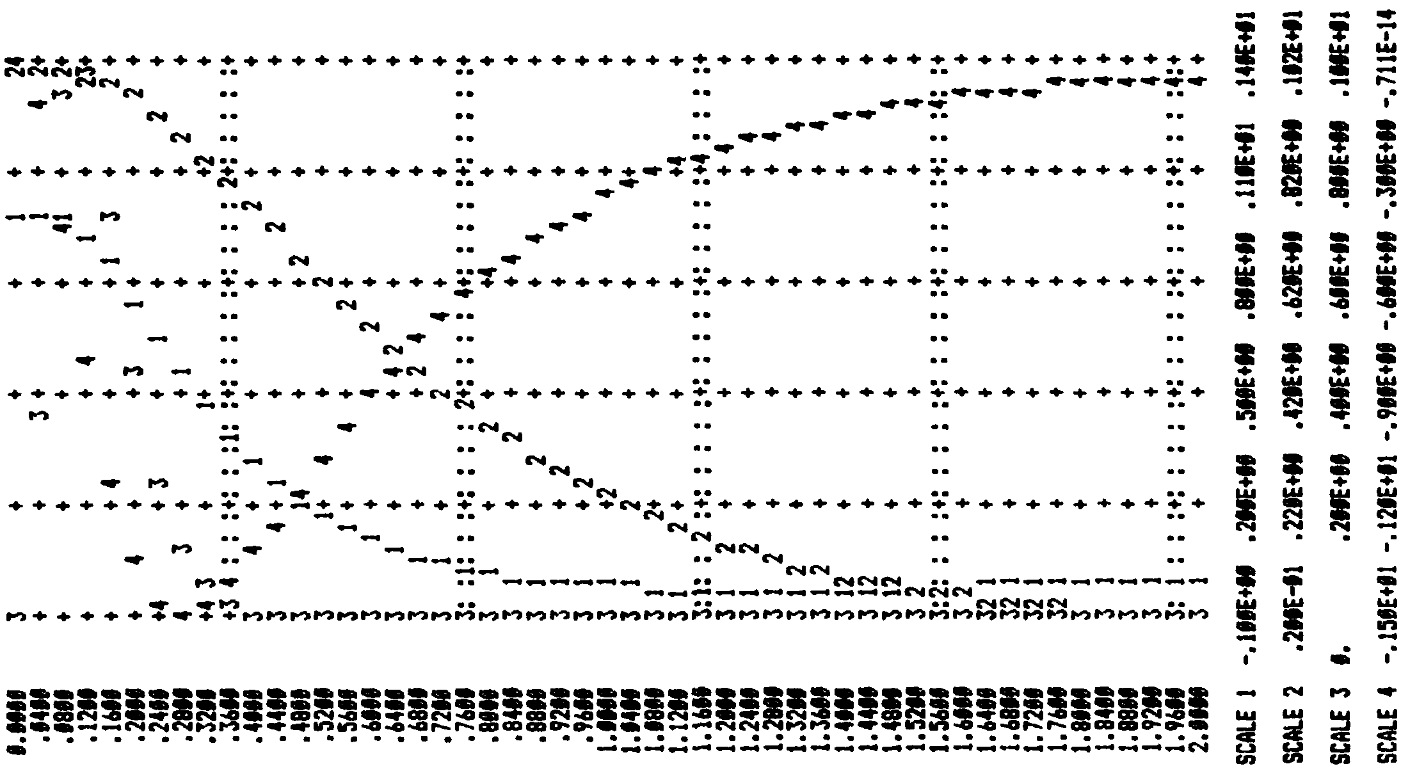


Figure V-26. Design IMF3-1, Nonlinear Single-State Actuators, Initial Condition = 1 Degree

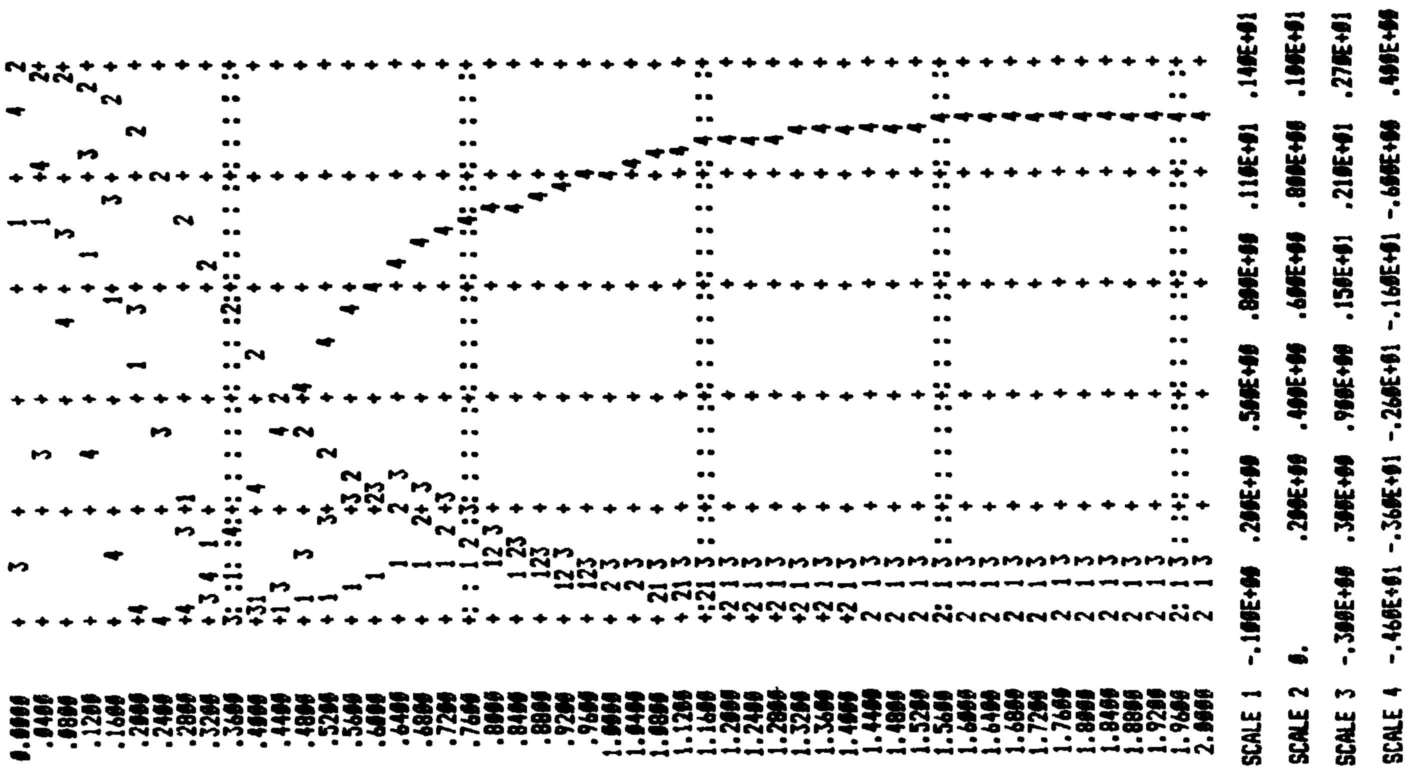


Figure V-27. Design IMF3-2, Nonlinear Single-State Actuators,
Initial Condition = 1 Degree

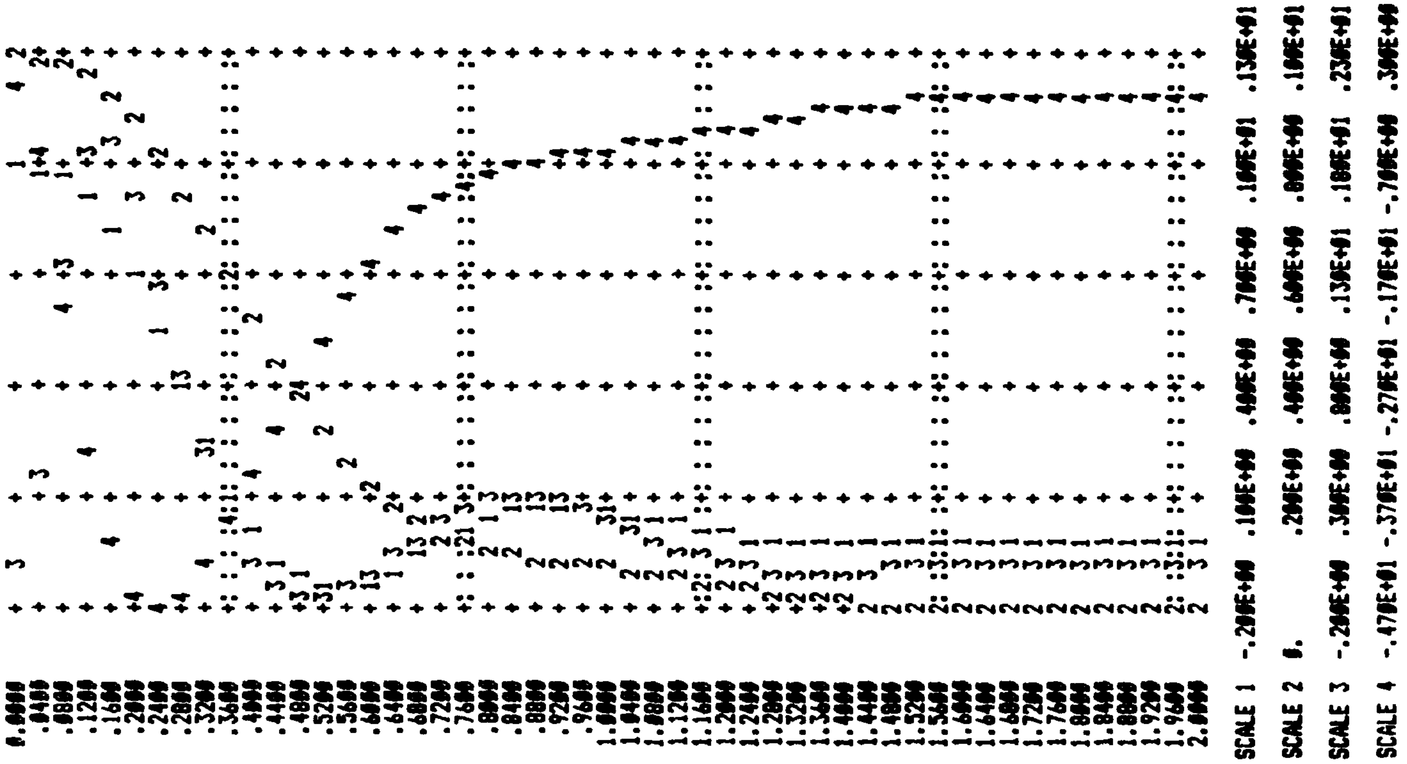


Figure V-28. Design IMF3-3, Nonlinear Single-State Actuators, Initial Condition = 1 Degree

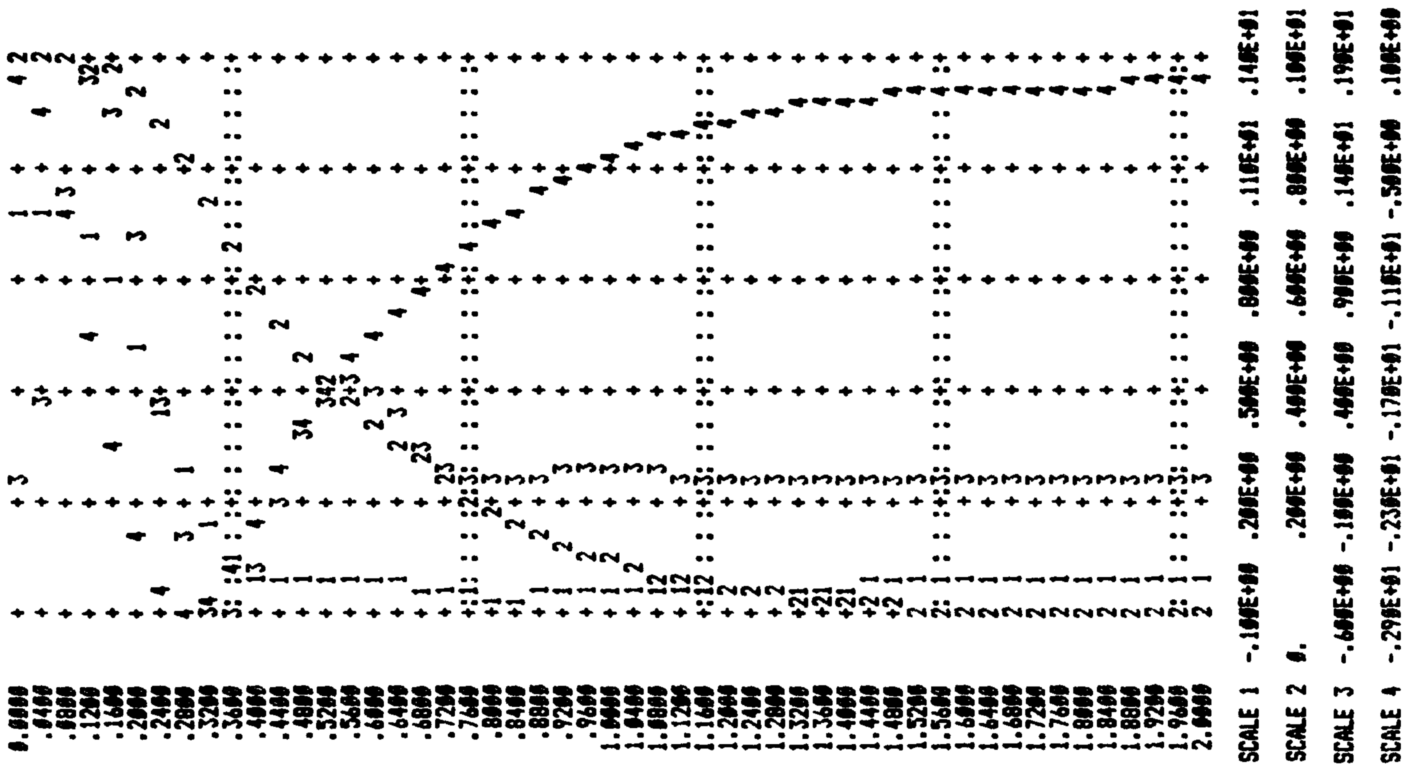


Figure V-29. Design IMF3-4, Nonlinear Single-State Actuators, Initial Condition = 1 Degree

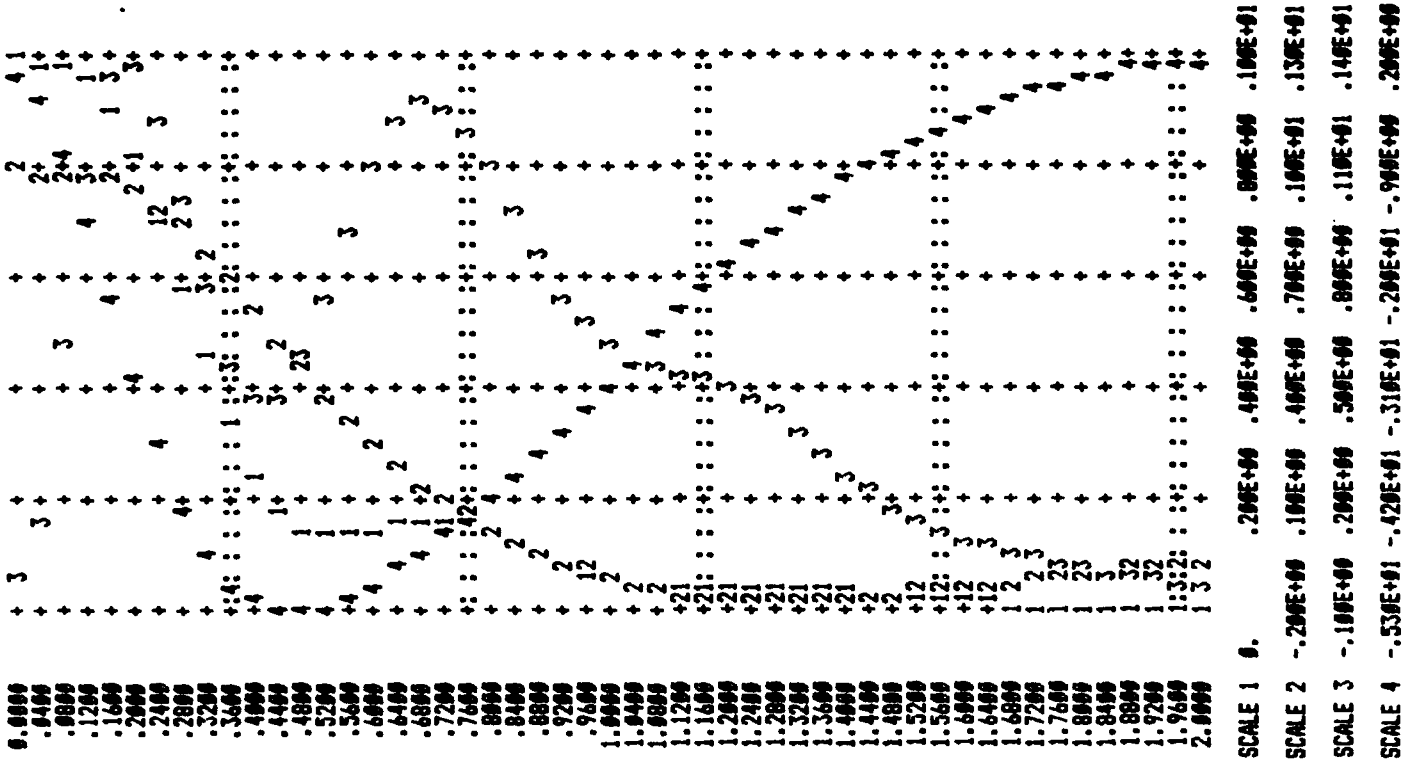


Figure V-30. Design IMF3-5, Nonlinear Single-State Actuators, Initial Condition = 1 Degree

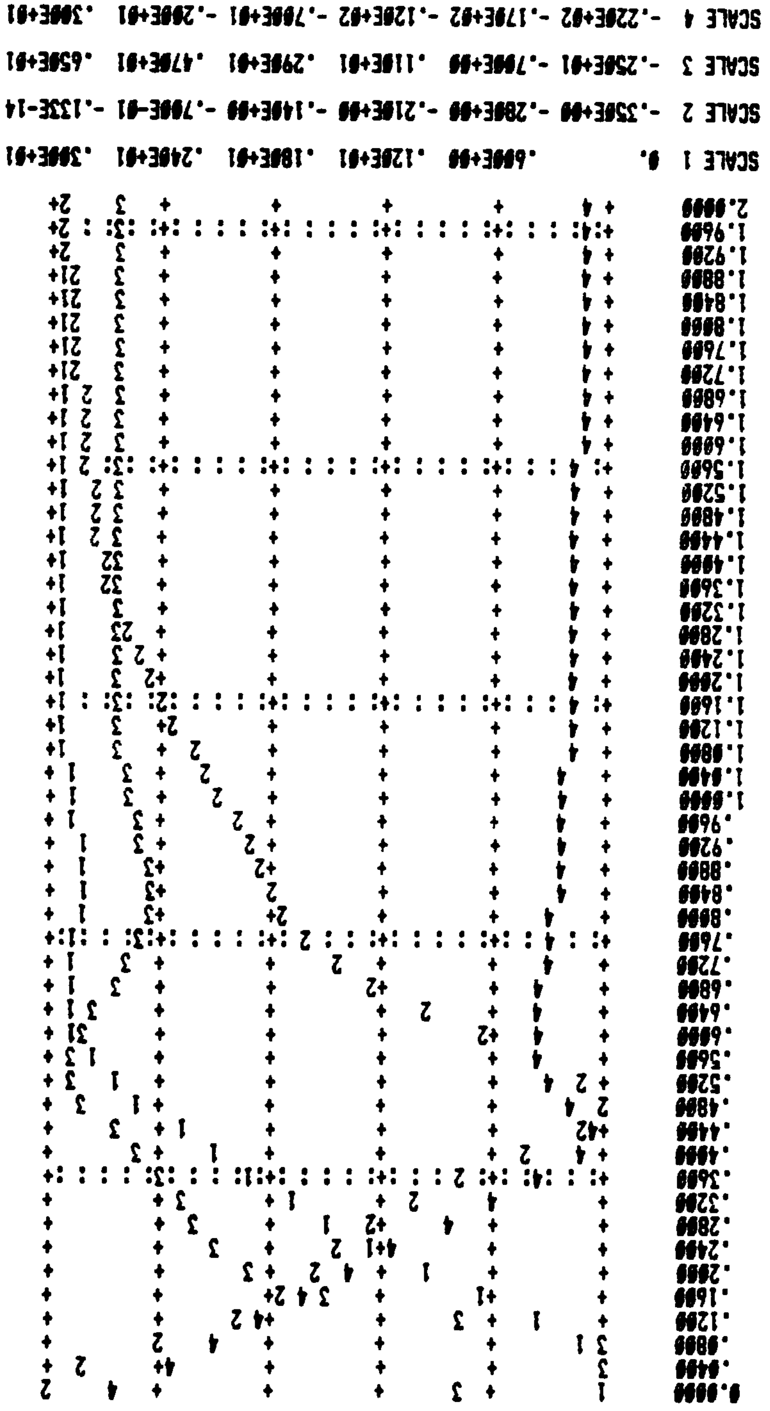


Figure V-31. Hybrid Design (PI Regulator IMF3-2, CGT from IMF2-5), Nonlinear Single-State Actuators, CGT Step Input = 3 Degrees

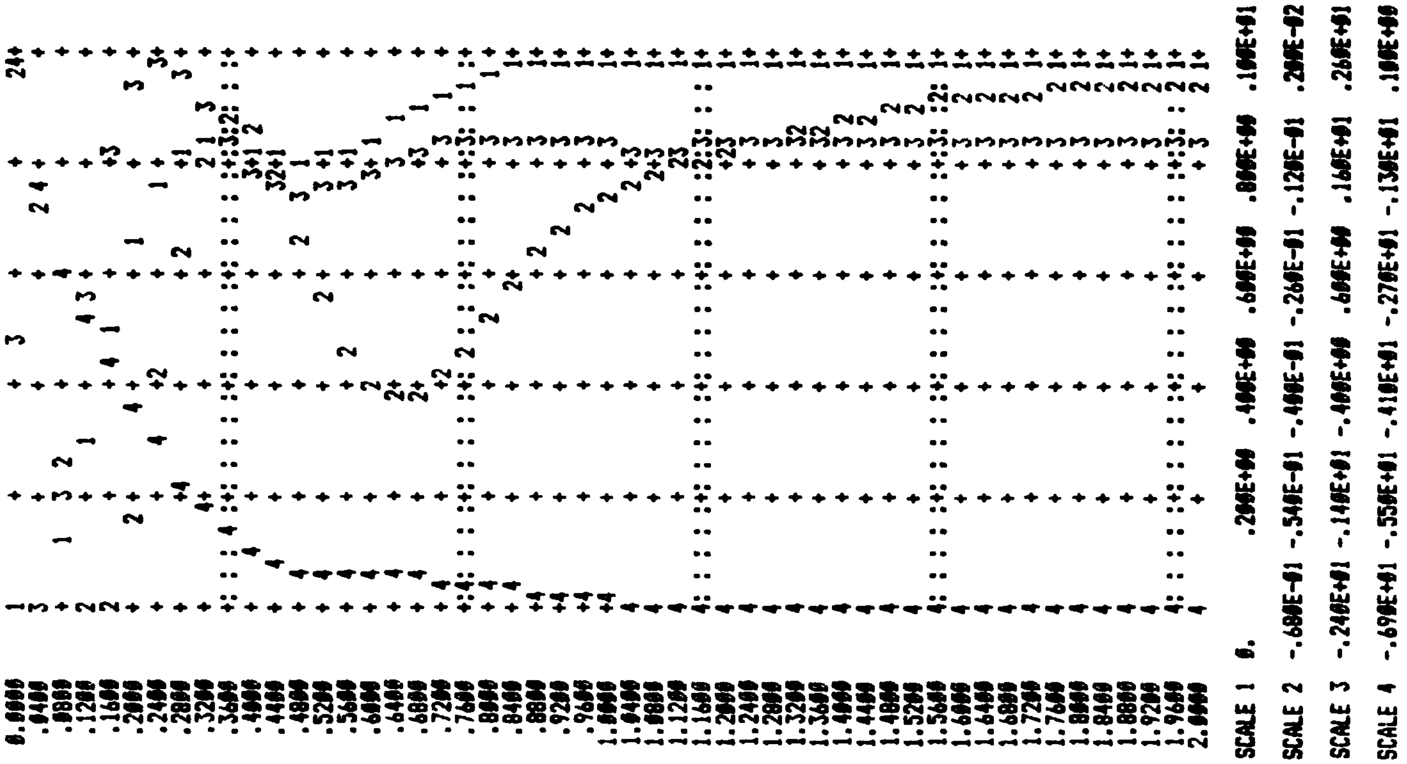


Figure V-32. Hybrid Design (PI Regulator IMF3-2, CGT from IMF2-2), Nonlinear Single-State Actuators, CGT Step Input = 1 Degree

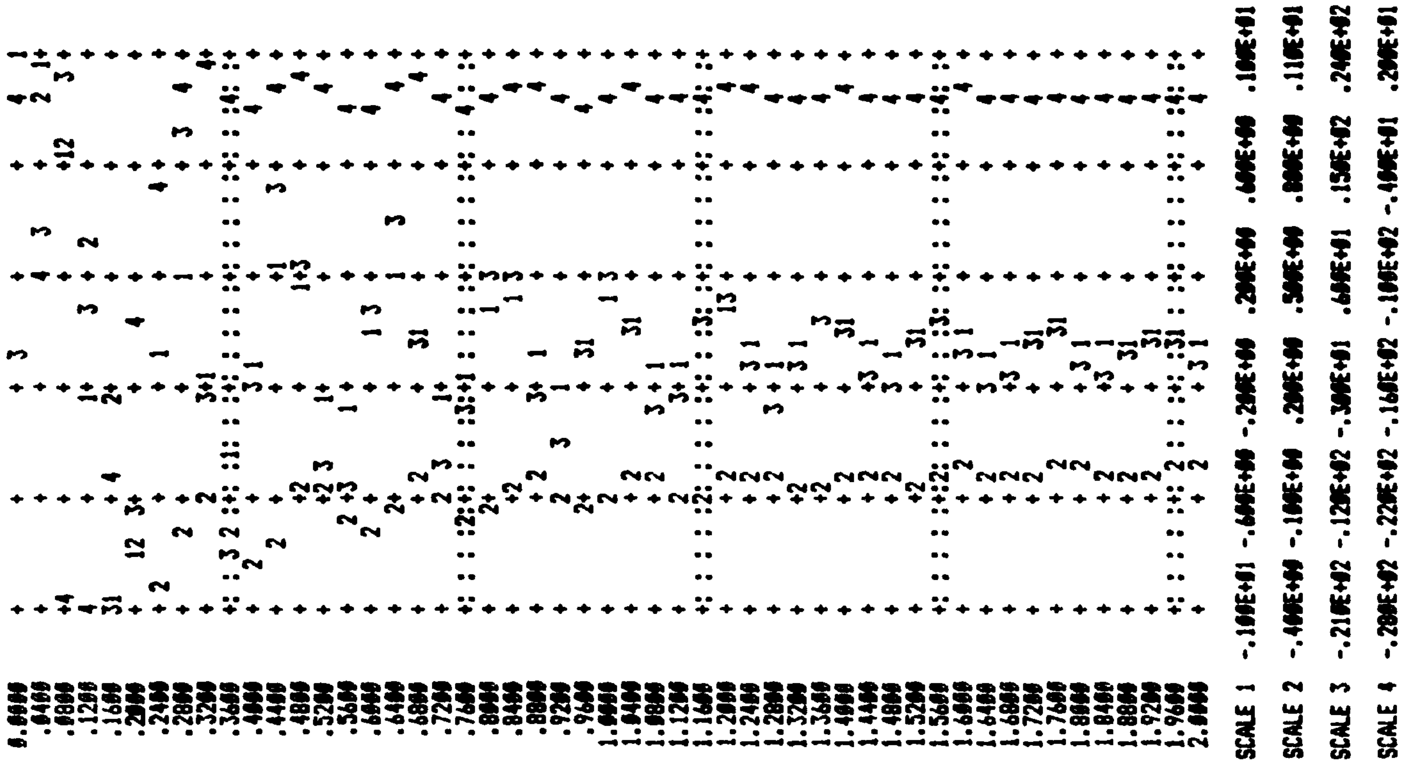


Figure V-33. Design SR-2, Linear Four-State Actuators, Initial Condition = 1 Degree

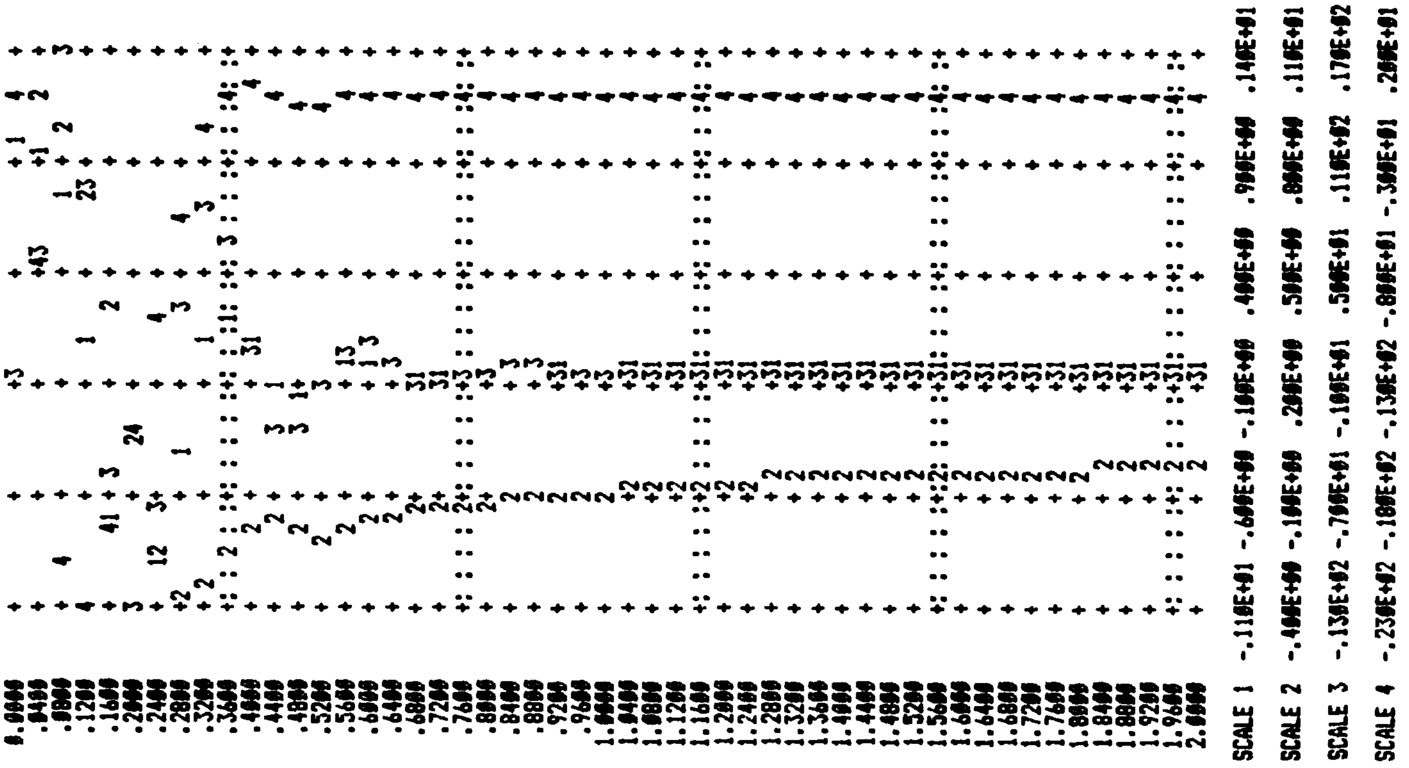


Figure V-34. Design SR-3, Linear Four-State Actuators, Initial Condition = 1 Degree

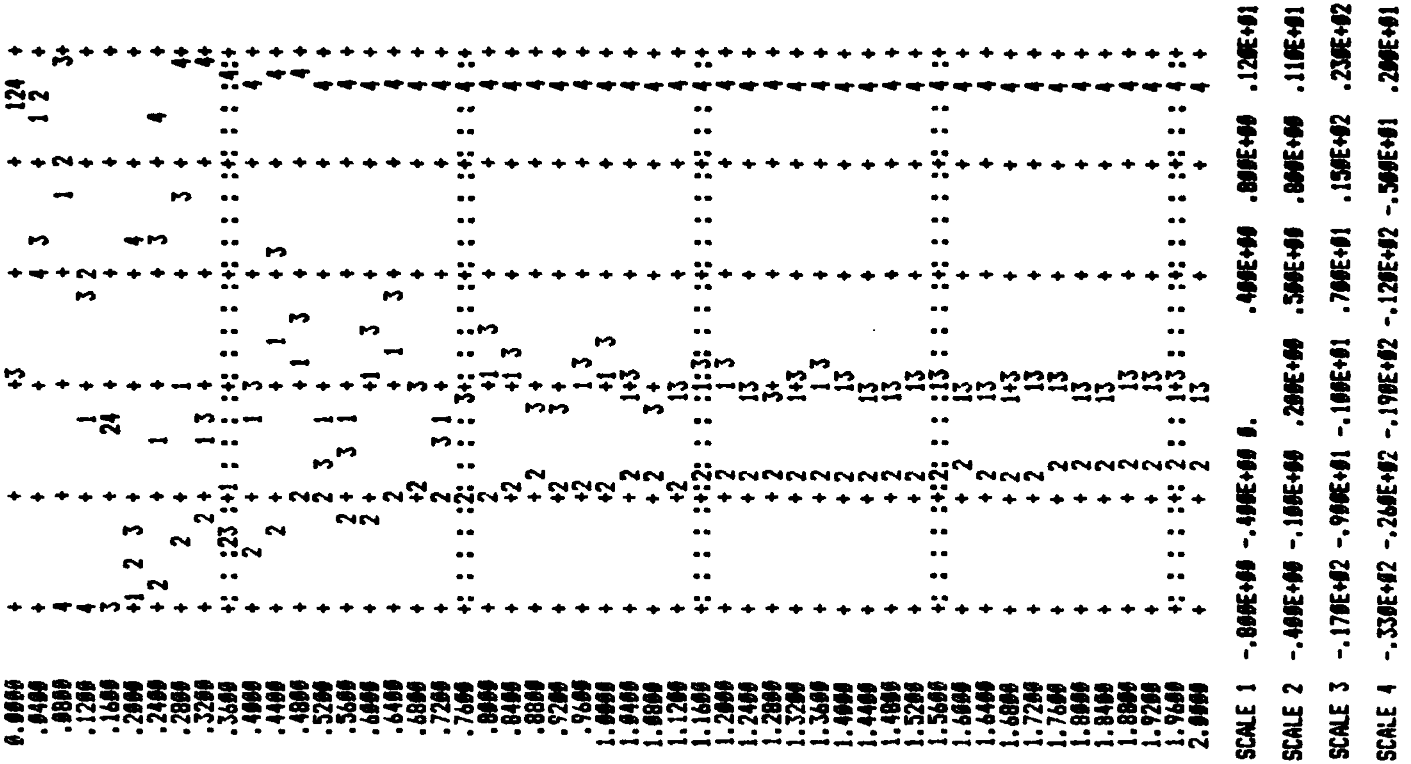


Figure V-35. Design IMF2-1, Linear Four-State Actuators, Initial Condition = 1 Degree

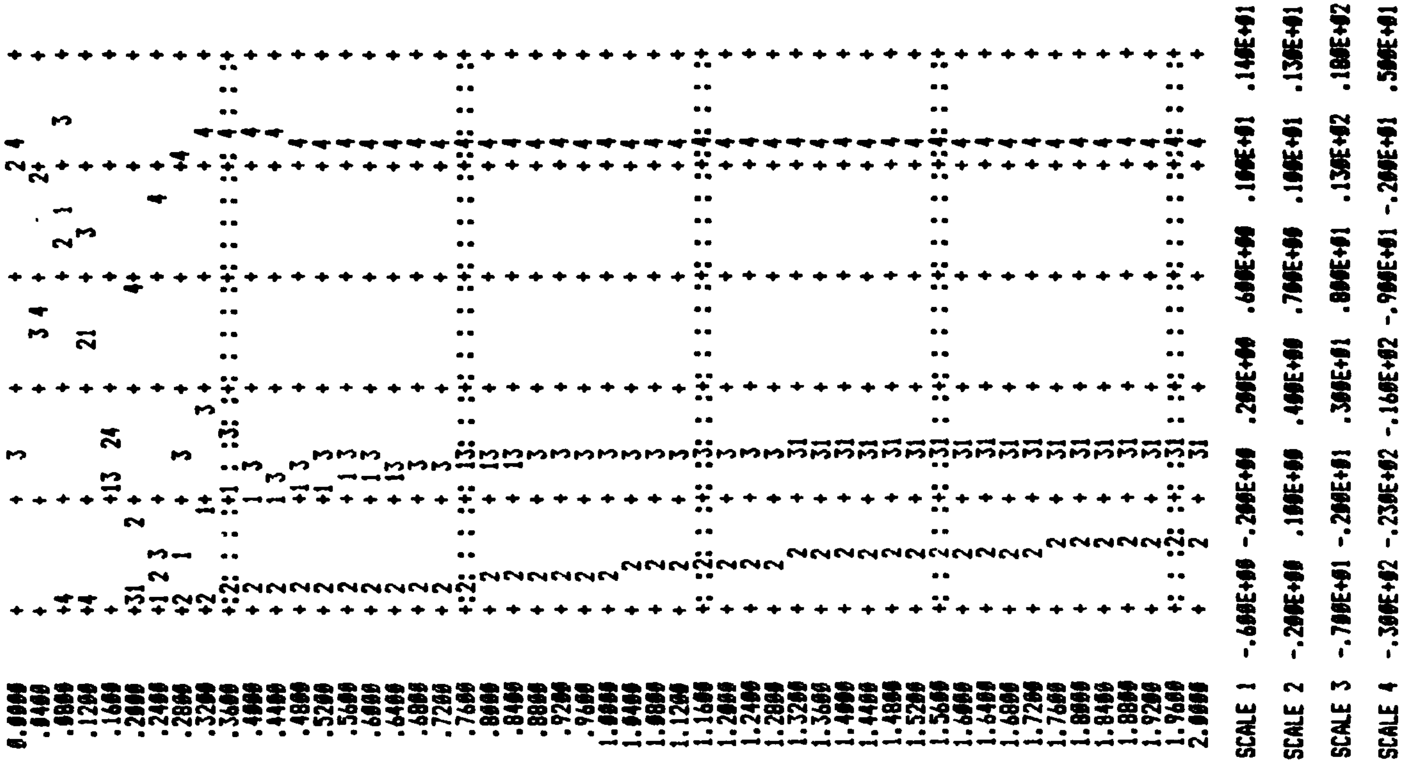


Figure V-36. Design IMF2-3, Linear Four-State Actuators, Initial Condition = 1 Degree

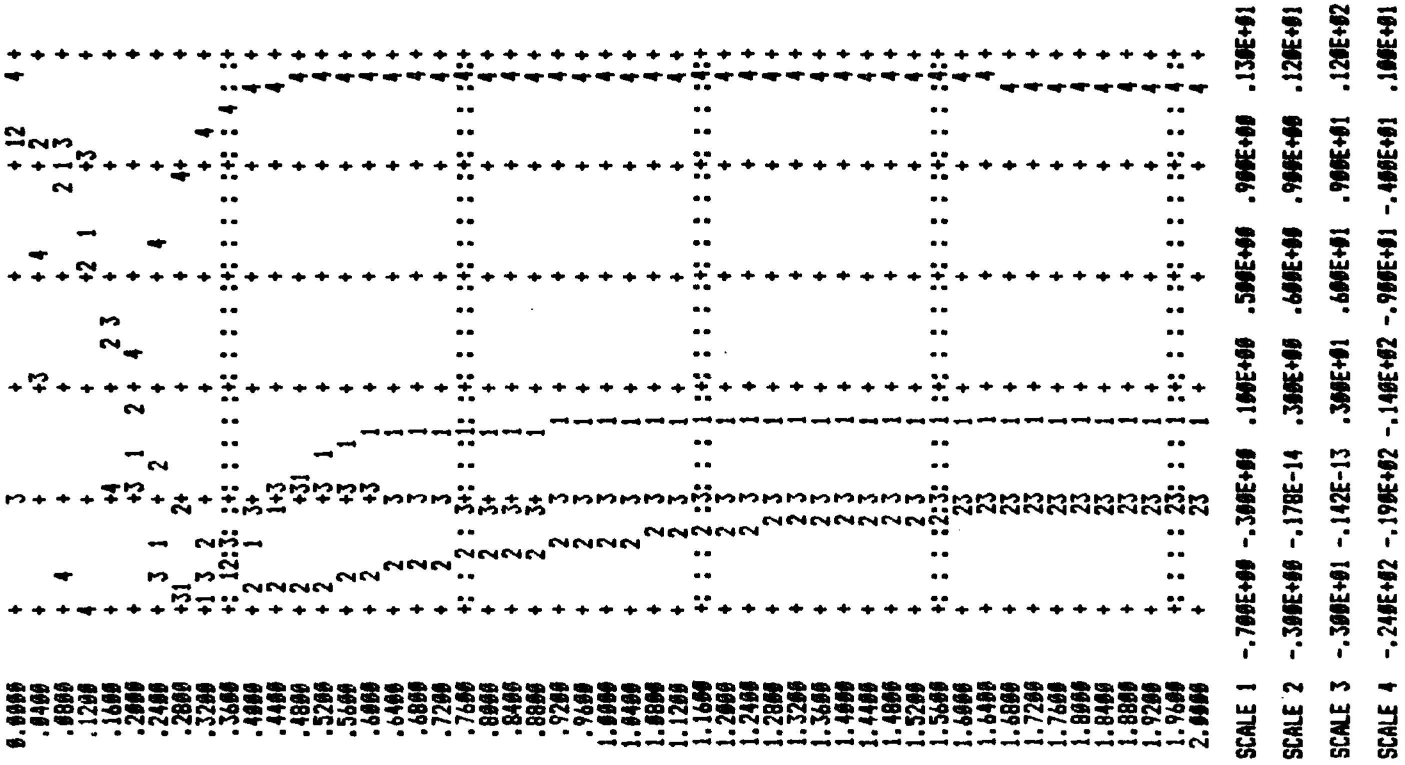


Figure V-37. Design IMF2-4, Linear Four-State Actuators, Initial Condition = 1 Degree

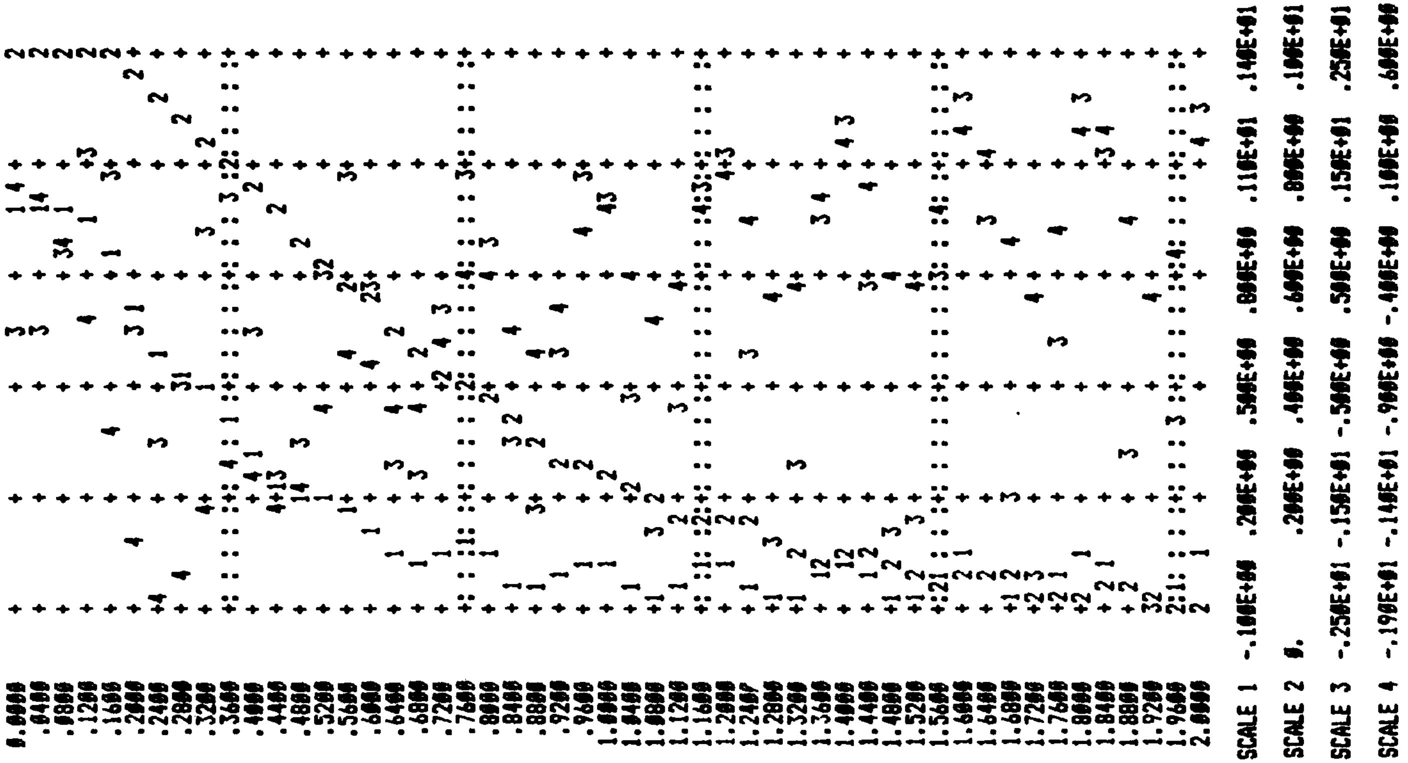


Figure V-38. Design IMF3-1, Linear Four-State Actuators, Initial Condition = 1 Degree

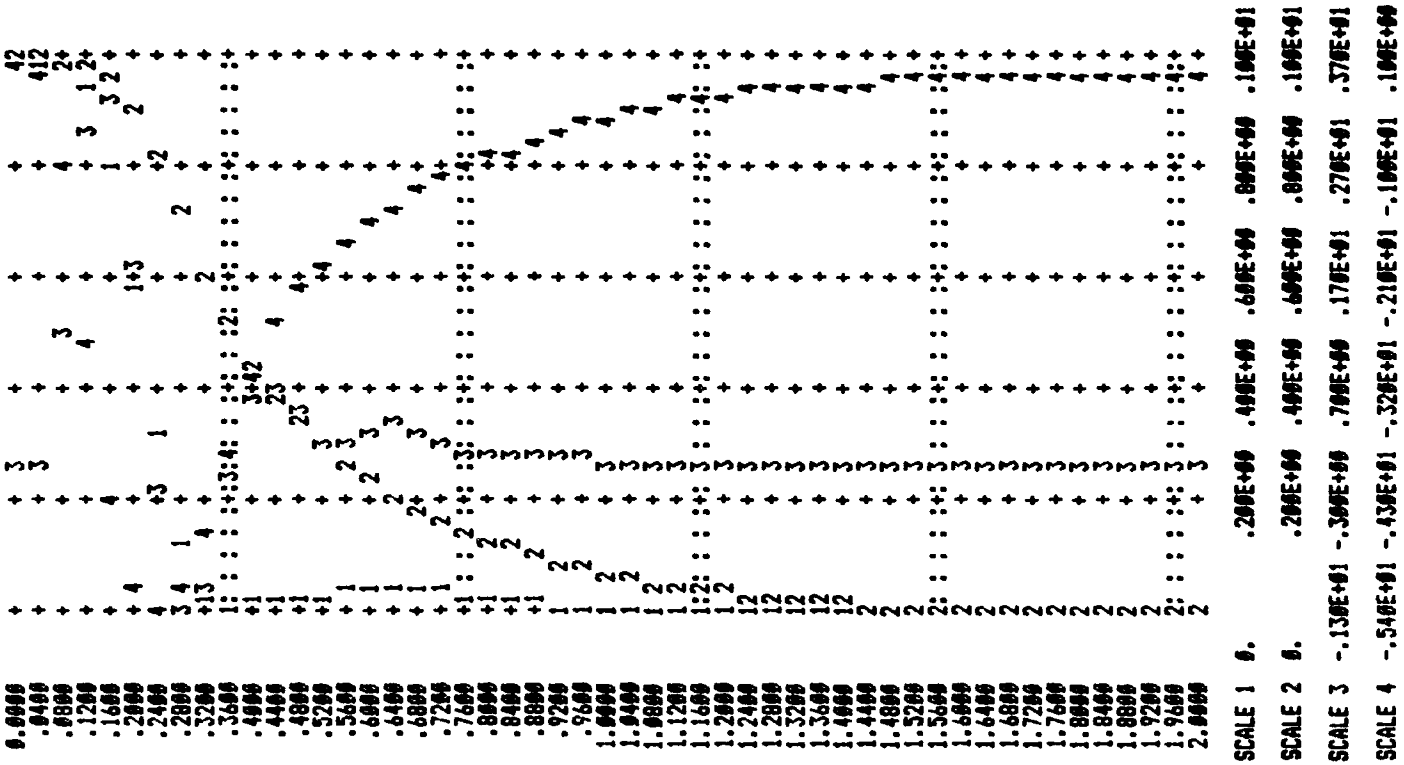


Figure V-39. Design IMF3-2, Linear Four-State Actuators, Initial Condition = 1 Degree

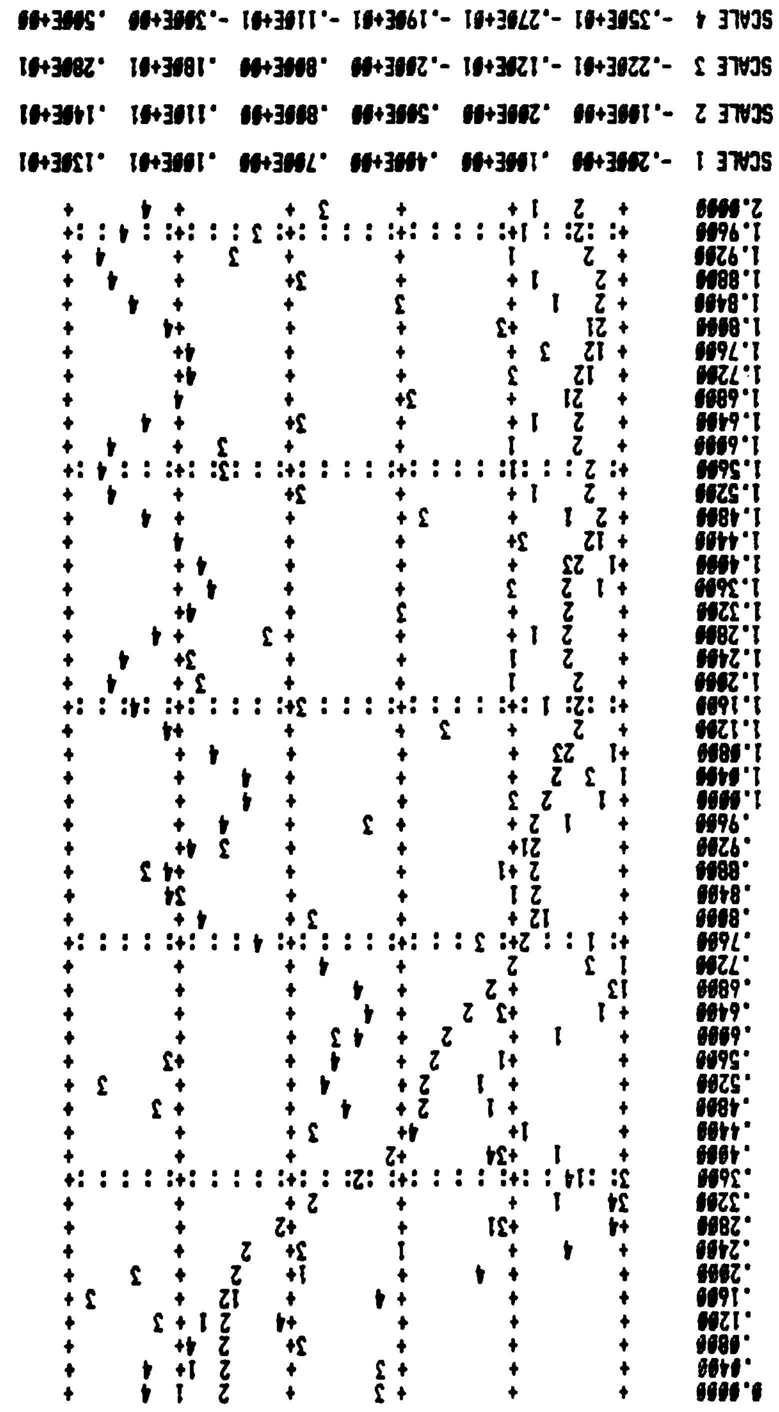


Figure V-40. Design IMF3-4, Linear Four-State Actuators, Initial Condition = 1 Degree

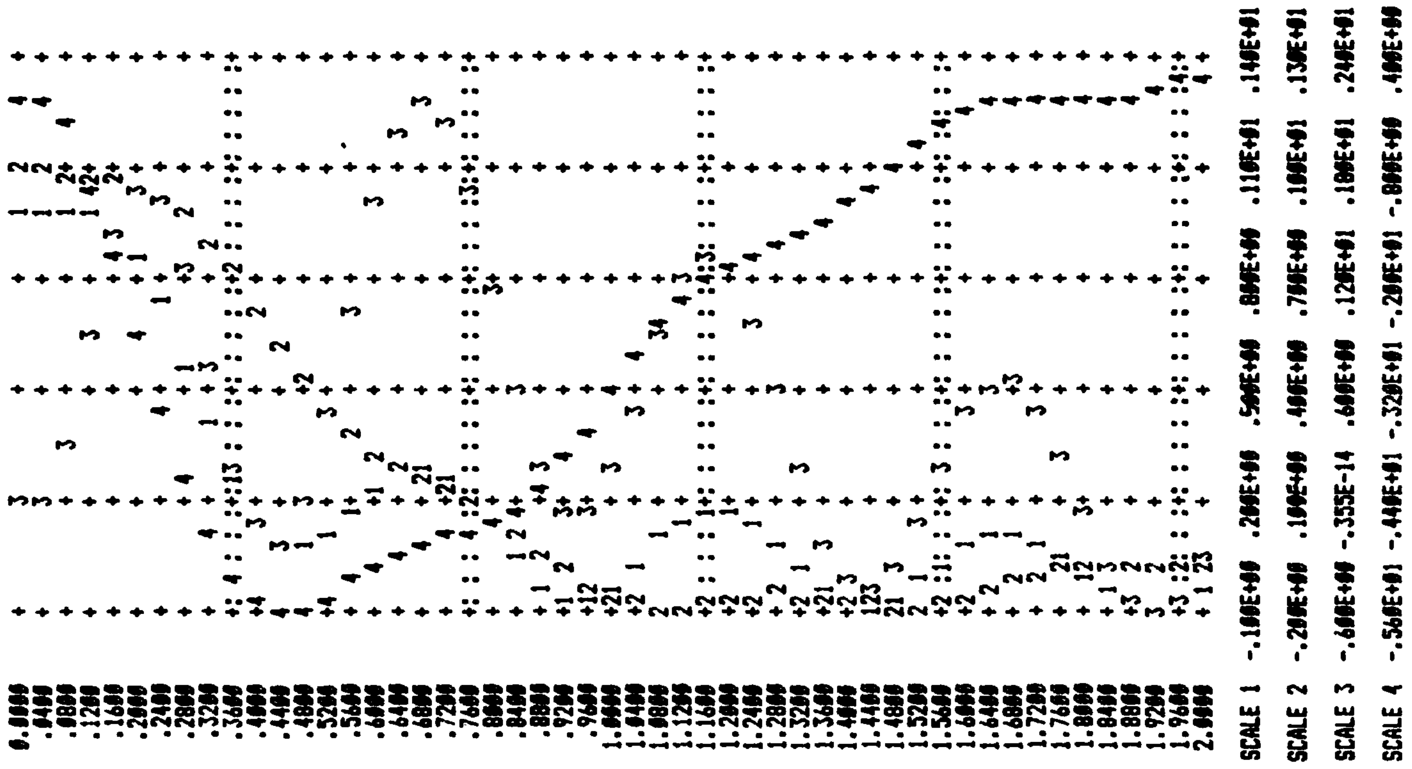


Figure V-41. Design IMF3-5, Linear Four-State Actuators, Initial Condition = 1 Degree

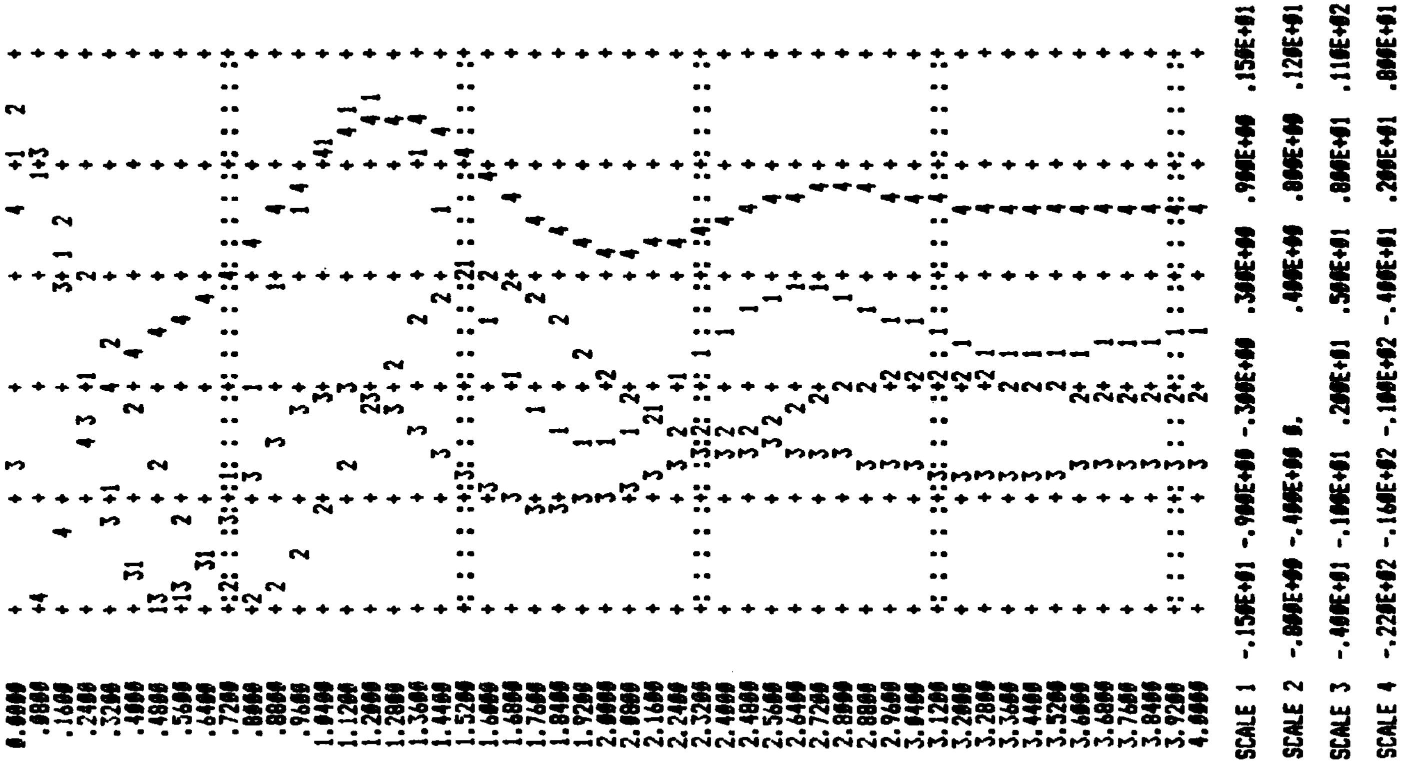


Figure V-42. Design IMF2-4, Linear Single-State Actuators, Flight Condition = .6 Mach/20,000 feet, Initial Condition = 1 Degree

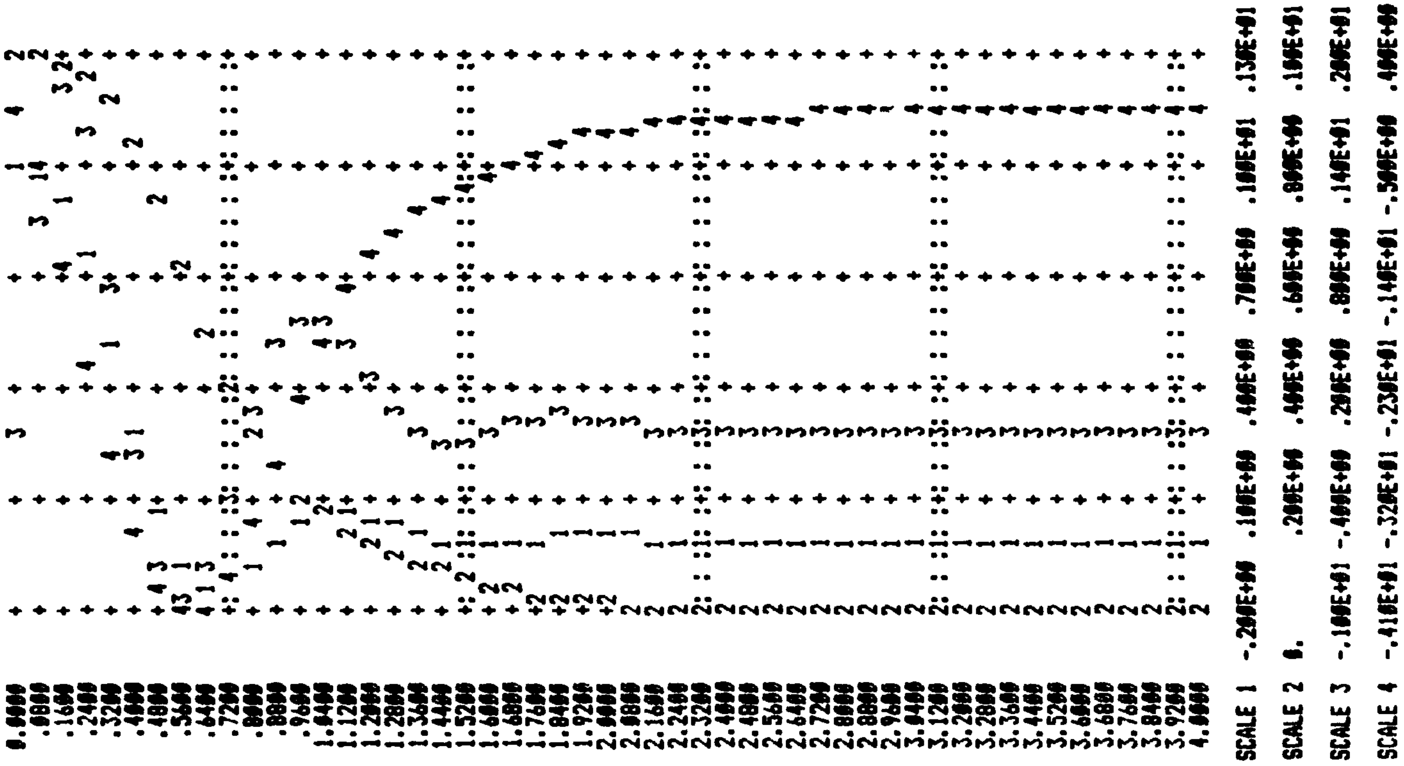


Figure V-43. Design IMF3-1, Linear Single-State Actuators, Flight Condition = .6 Mach/20,000 feet, Initial Condition = 1 Degree

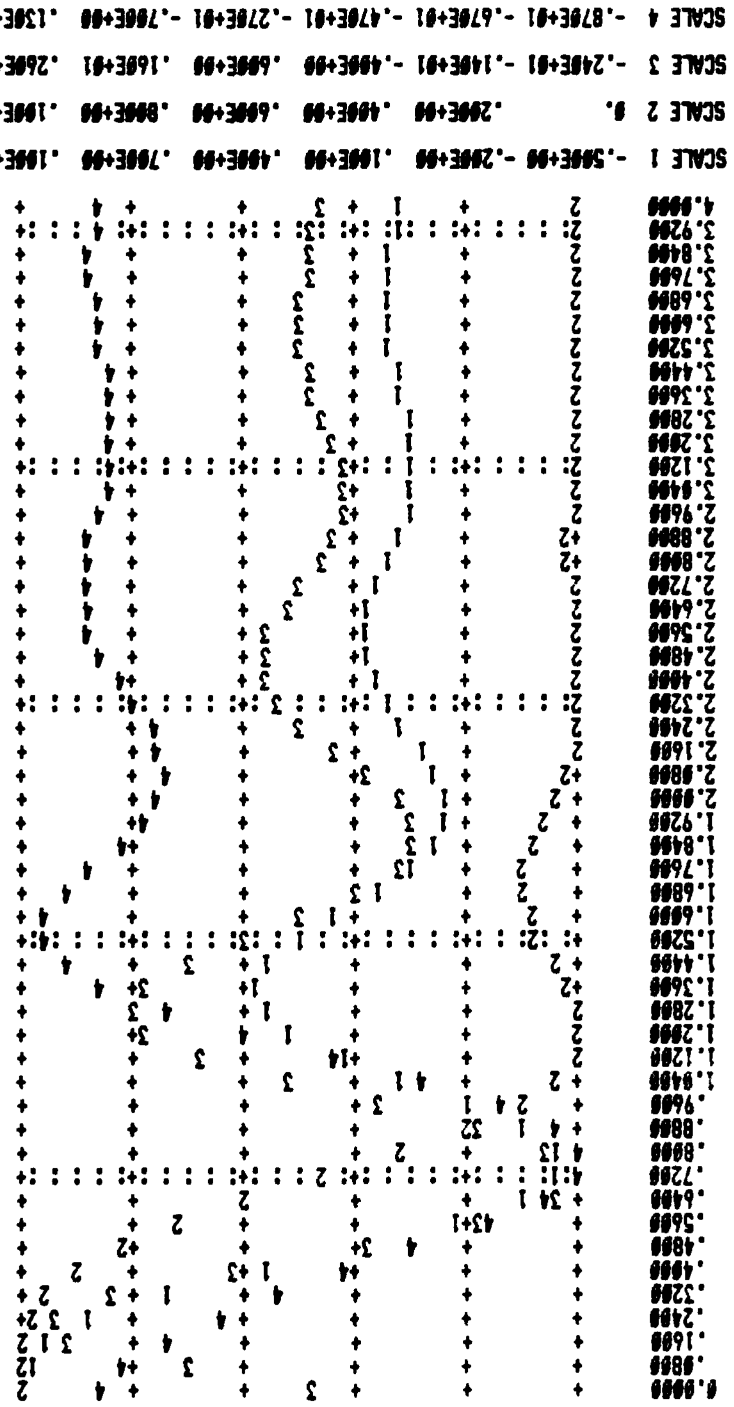


Figure V-44. Design IMF3-1, Linear Single-State Actuators, Flight Condition = .6 Mach/30,000 feet, Initial Condition = 1 Degree

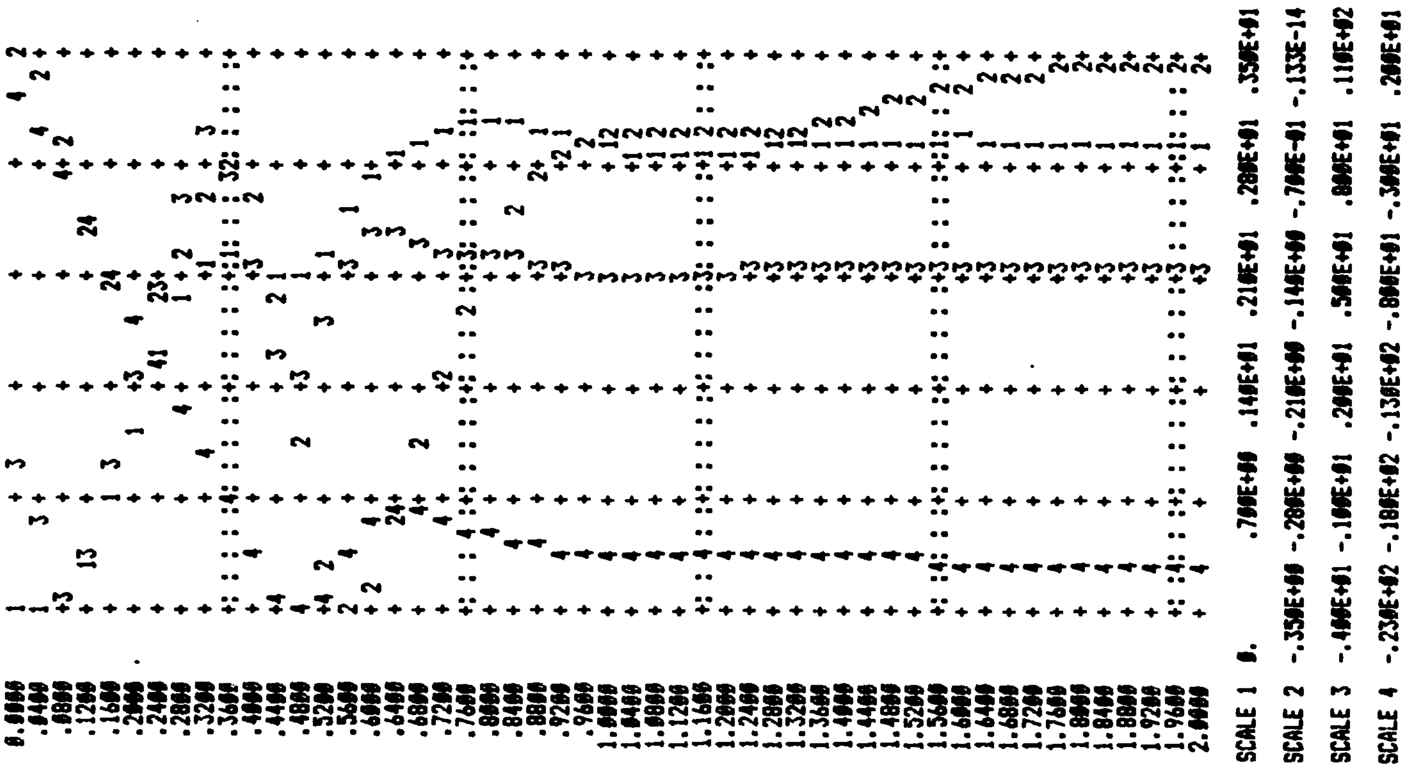


Figure V-45. Hybrid Design (PI Regulator IMF3-3, Undesignated CGT), Nonlinear Four-State Actuators, CGT Step Input = 3 Degrees

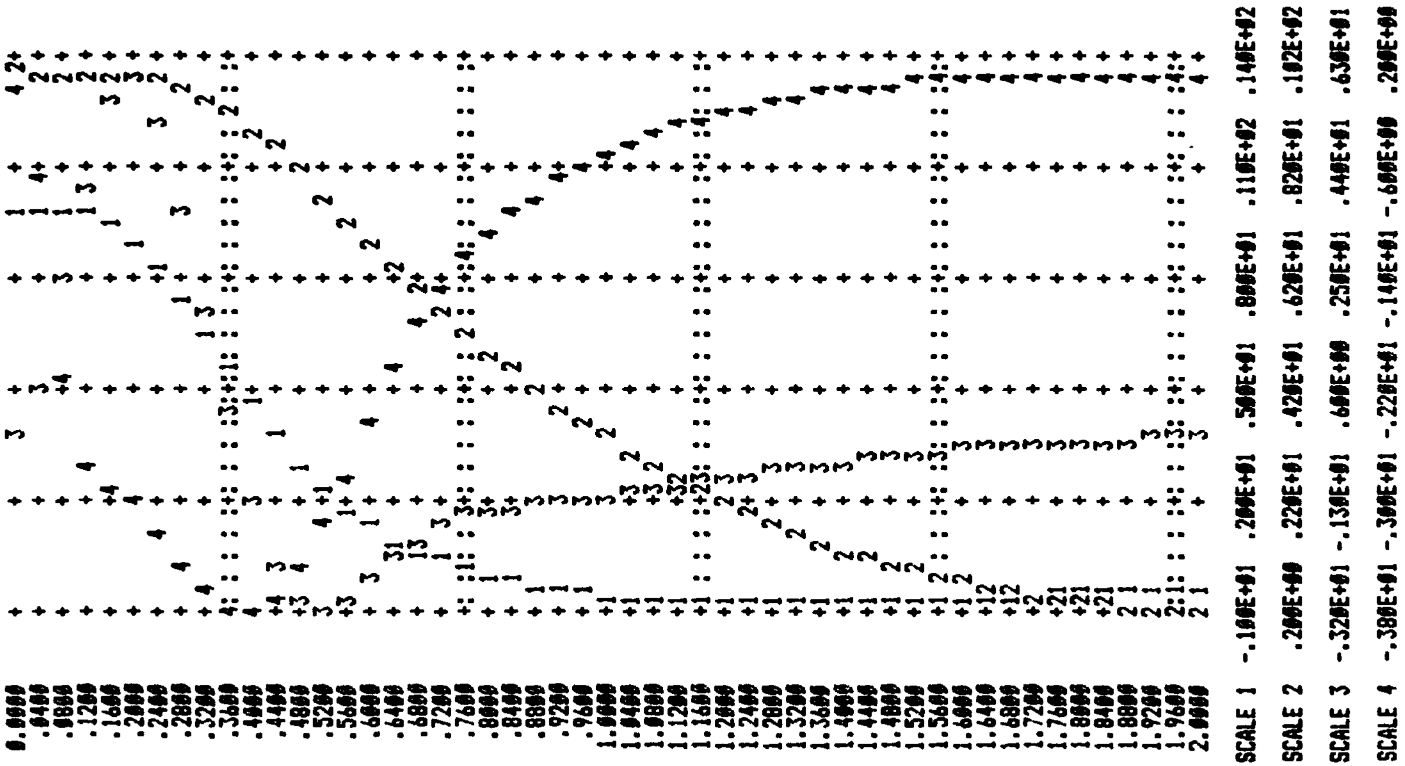


Figure V-46. Design IMF2-4, Nonlinear Four-State Actuators, Anti-Windup Compensated, Initial Condition = 10 Degrees

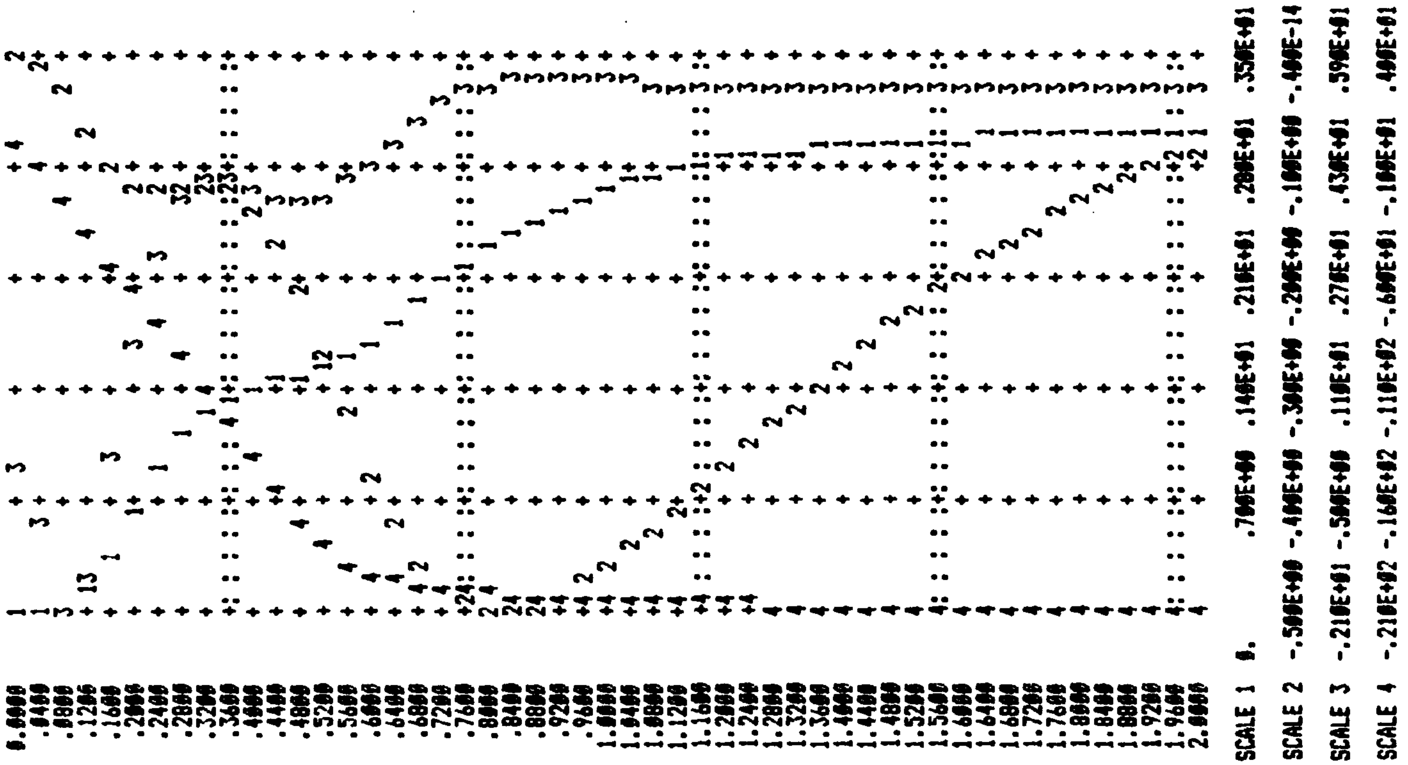


Figure V-47. Design IMF2-4, Nonlinear Four-State Actuators, Anti-Windup Compensated, CGT Step Input = 3 Degrees

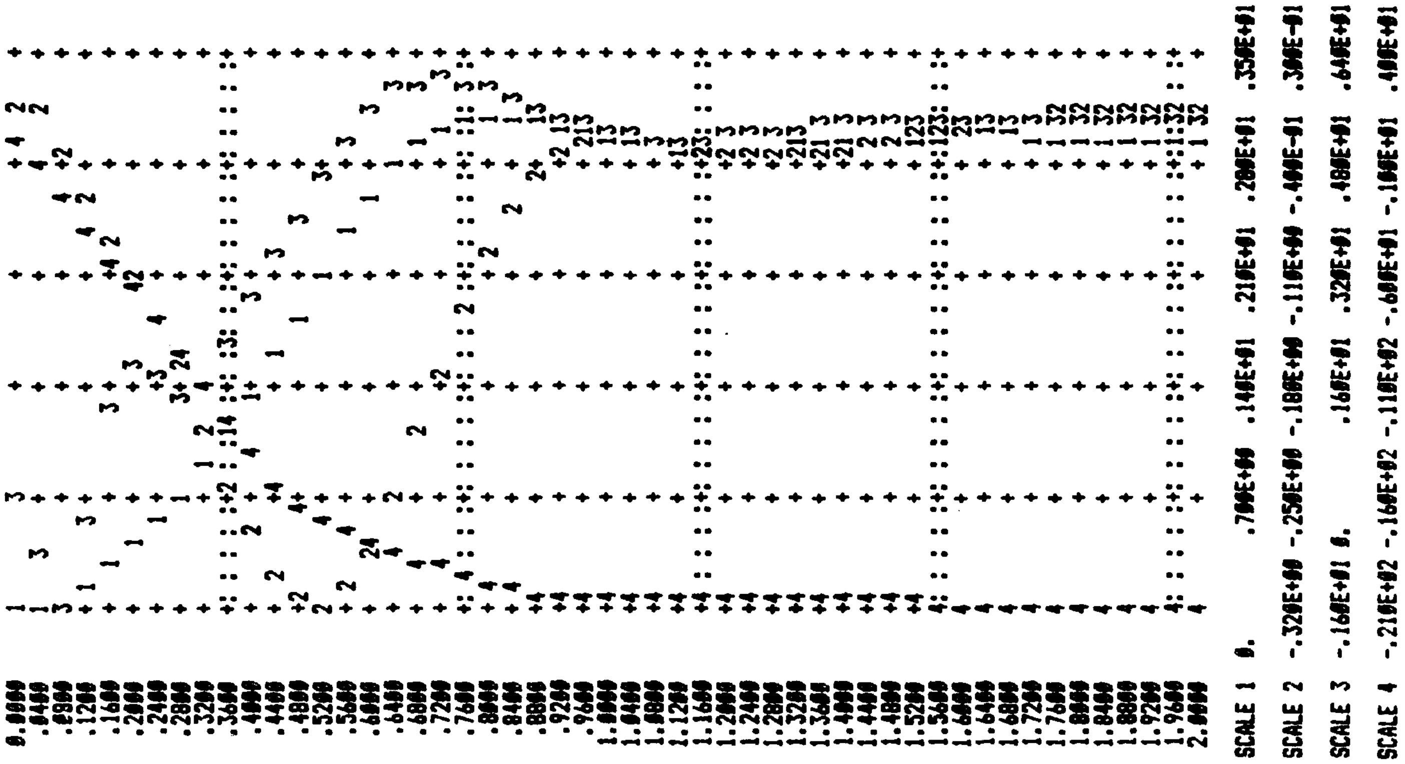


Figure V-48. Hybrid Design (PI Regulator IMF3-3, Undesignated CGT), Nonlinear Four-State Actuators, Anti-Windup Compensated, CGT Step Input = 3 Degrees

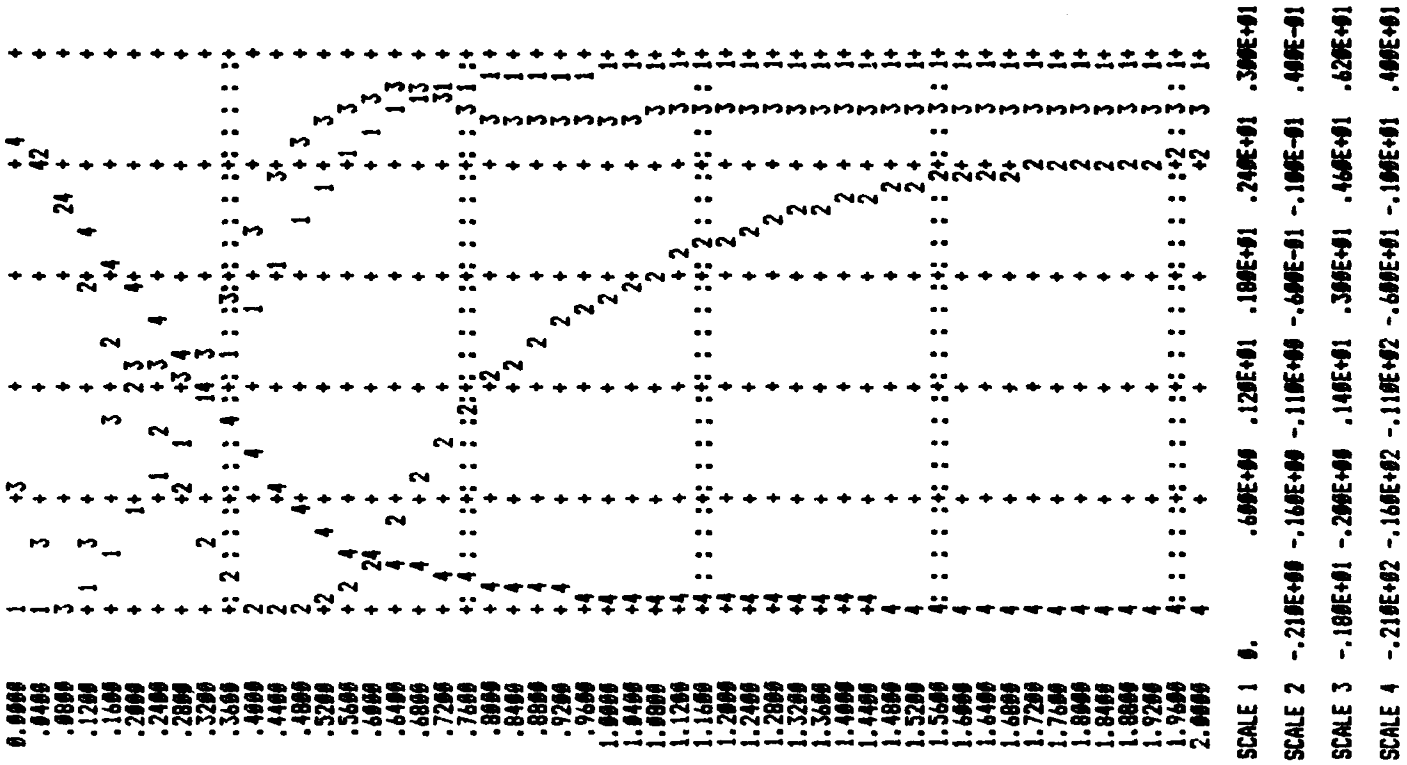


Figure V-49. Hybrid Design (PI Regulator IMF3-2, CGT from IMF2-5), Nonlinear Four-State Actuators, Anti-Windup Compensated, CGT Step Input = 3 Degrees

VI. Conclusions and Recommendations

6.1 Implicit Model-Following

This study has demonstrated that the incorporation of an implicit model into the performance index used to define a linear-quadratic PI regulator can provide greater control over many aspects of the resulting controller design than that achievable using a "standard" PI regulator. In particular, implicit model-following was shown to be useful in controlling the speed of the initial response of the regulator, as characterized by rise time. This produced a controller which could perform acceptably, despite the constraints of harsh nonlinearities inherent in the controlled system, because it was designed so that it could perform its function without exceeding the range of linear operation. Compared to design methods using conventional PI regulator formulations, implicit model-following was shown to provide an increased ability to control the locations of the closed-loop system's poles, in a systematic and easily predictable fashion, by means of iterative changes to the quadratic weightings used in the design. This ability was found to be useful in designing to avoid the effects of higher-order dynamics inherent in the controlled system, but ignored in the design model.

The use of an "all-implicit" PI regulator, based only on the quadratic cost of (A-43), was found to be feasible. Good transient response and steady-state regulation were both achievable without the use of "explicit" weights applied directly to the output deviations.

For the designs presented in this thesis, it appeared that achieving the capability to operate well despite one type of perturbation generally occurred at the expense of robustness against another type. The designs conducted with the standard regulator formulation exhibited relatively good performance with regard to parameter variations, but were degraded by higher-order dynamics and actuator nonlinearities. Implicit designs developed to improve robustness with respect to higher-order dynamics and nonlinearities were generally somewhat less robust with regard to parameter variations. This highlighted the need for some insight on the part of the designer as to the types of uncertainty prevalent in the design model.

An ability to provide a general increase in robustness by forcing the closed-loop system to have maximally orthogonal eigenvectors was not shown. The robustness benefits of such an eigenstructure have been documented [19]. However, the closed-loop PI regulator formulation is fairly complex; the number and dimension of its eigenvectors are large in comparison to the number of controls available to affect their form. Admittedly, the eigenvectors of the designs were not calculated in this study, and the design iterations were focused on achieving specific objectives other than an orthogonal eigenstructure. It is quite possible that implicit model-following might be shown to have potential general robustness benefits, at least for some problems, if an effort were made to design specifically toward achieving maximally orthogonal eigenvectors. Such a design approach could inevitably be expected to require some sort of tradeoff with other characteristics of the resulting design, such as performance capabilities.

6.2 Robustness Analysis

In this study, "unstructured" singular value analysis was found to be of limited value in predicting how well a system would perform with respect to off-nominal conditions. There are several factors which contributed to this observation.

The first factor is the variable conservatism of robustness estimates that are based on this type of singular value analysis. Even if the norm of a particular perturbation were known exactly, comparison to the minimum singular value of the closed-loop system's inverse return difference function could provide, at best, only positive proof that the perturbation could not be destabilizing. If such an analysis were to show the norm of the perturbation to be sufficient to cause instability, there would remain three possibilities:

-- That instability would result.

-- That performance would be degraded, but the system would be stable.

-- That little or no degradation to performance or stability would result.

Some other method of analysis, such as simulation, would be required to determine the actual effect of such a perturbation.

The second, and related, factor is that this type of analysis presupposes that the designer can formulate an estimate of the norm of the uncertainties that should be guarded against. As pointed out in Section 5.6, the norm of the multiplicative perturbation that results from a change to the nominal dynamics description is, in fact, a fairly complex function of both that change and the control law. Consideration of the effects of nonlinearities presents a particularly difficult problem. This adds to the uncertainty as to which, if any, possibly destabilizing perturbations would actually produce instability. Design iterations aimed simply at increasing minimum singular values could, in fact, result in trading off performance to guard against a perturbation which could not occur physically.

The final factor is the fact that, in many cases, simulation can readily provide the type of robustness information that the designer needs, in a form which is unambiguous and easy to interpret. While it is impossible to simulate all of the possible perturbations, judicious selection of example cases which encompass the spectrum of known areas of uncertainty is possible. Nonlinearities, shown in this study to be of significant potential impact, are easy to simulate, and the type of simulation used for the deterministic controllers in this study can easily be extended to the Monte Carlo analysis of stochastic controllers [29,36]. Sensitivity analysis can be conducted in a straightforward way, through simulation, to gain insight into the physical factors exercising the greatest influence over performance and stability.

The results of the singular value analysis conducted in this study were not, in general, inconsistent with the simulation results; they simply added little insight. In some cases, the singular value analysis results were very misleading, such as the relatively optimistic high frequency robustness predictions that could have been made for designs IMF3-4 and IMF3-5.

6.3 Recommendations for Further Study

A great deal of the design effort described in this thesis centered on combatting an effect called "windup" which occurs in PI regulators, due to control actuator saturation. A concurrent thesis effort by Lt James McMillian [32] has produced another modification to the CGT/PI/KF design software that implements a generic formulation of the PI regulator [30]. In this implementation, the proportional and integral channels of the regulator output feedback are distinct, and need not be equal. The susceptibility of a PI regulator to the windup phenomenon should be a function of the magnitude of the integral channel feedback relative to that of the proportional channel. Anti-windup compensation through the use of the capabilities provided by Lt McMillian's program, in conjunction with implicit model-following, should be investigated. The effects of such compensation on the general robustness of the controller should also be determined.

As discussed in Section 5.5, use of a diagonal regulator command model dynamics matrix in this study represented only one of many possible command model structures. Investigation of the performance and robustness characteristics of implicit regulator designs based on different structures, i.e., with various types and numbers of off-diagonal command model dynamics matrix elements, should be conducted.

For the designs of this study, the use of unstructured singular value analysis was found to provide little predictive insight into design robustness. As mentioned in Section 3.2, a great deal of current study is underway involving the use of "structured" singular value analysis [12,15,18,27]. To employ such sophisticated methods, the designer must have insight into the structure of the types of uncertainty that are likely to be present in the control system. Such insight was not in the repertoire of the author of this thesis. A study of methods of analyzing the structure of modelling error and the application of knowledge of that structure to singular value analysis is a recommended area for future research.

Another concurrent thesis effort by Lt Jean Howey [22] has centered on the recovery of robustness lost due to the employment of a Kalman filter to provide state estimates for a PI regulator. The effectiveness of the methods developed in that study should be determined for control systems in which implicit model-following is employed, as in this research. A logical extension of such a study would include modelling and analyzing the complete system with a human operator in the loop.

Bibliography

1. Andry, A. N., Jr., E. Y. Shapiro and J. C. Chung. "On Eigenstructure Assignment for Linear Systems." Unpublished paper. Lockheed California Company, Burbank, California. Undated.

2. Barfield, A. F. Flight Control Engineer (personal conversation). Air Force Wright Aeronautical Laboratories, Flight Dynamics Laboratory, Wright-Patterson Air Force Base, Ohio. 10 July - 10 October 1983.

3. Barraud, A. Y. "A Numerical Algorithm to Solve A XX-X=Q," IEEE Transactions on Automatic Control, AC-22 (5): 883-885 (October 1977).

4. Barrett, M. F. "Conservatism with Robustness Tests for Linear Feedback Control Systems," Proceedings of the 19th IEEE Conference on Decision and Control, Albuquerque, New Mexico. 885-890 (December 1980).

5. Broussard, J. R. "Command Generator Tracking," TASC TIM-612-3, The Analytical Sciences Corporation, Reading, Massachusetts, March 1978.

6. Broussard. J. R. and P. W. Berry. "The Relationship Between Implicit Model Following and Eigenvalue Eigenvector Placement," IEEE Transactions on Automatic Control, AC-25 (3): 591-594 (June 1980).

7. Cruz, J. B., J. S. Freudenberg and D. P. Looze. "A Relationship Between Sensitivity and Stability of Multivariable Feedback Systems," IEEE Transactions on Automatic Control, AC-26 (1): 66-74 (February 1981).

8. D'Azzo, J. J. and C. H. Houpis. Linear Control System Analysis and Design (Second Edition). New York: McGraw-Hill Book Company, 1981.

9. Dongarra, J. J., et al. LINPACK Users' Guide. Philadelphia: SIAM, 1979.

10. Doyle, J. C. "Guaranteed Margins for LQG Regulators," IEEE Transactions on Automatic Control, AC-23 (4): 756-757 (August 1978).

11. -----, "Robustness of Multiloop Linear Feedback Systems," Proceedings of the 1978 IEEE Conference on Decision and Control, San Diego, California. 12-18 (January 1979).

12. -----, "Analysis of Feedback Systems with Structured Uncertainties," IEE Proceedings, 129 (D,6): 242-250 (November 1982).

13. Doyle, J. C. and G. Stein. "Robustness with Observers," IEEE Transactions on Automatic Control, AC-24 (4): 607-611 (August 1979).

14. -----, "Multivariable Feedback Design: Concepts for a Classical/Modern Synthesis," IEEE Transactions on Automatic Control, AC 26 (1): 4-16 (February 1981).

15. Doyle, J. C., J. E. Wall and G. Stein. "Performance and Robustness Analysis for Structured Uncertainty," Proceedings of the 21st IEEE Conference on Decision and Control, Orlando, Florida. 629-636 (December 1982).

16. Floyd, R. M. Design of Advanced Digital Flight Control Systems Via Command Generator Tracker (CGT) Synthesis Techniques, Volumes 1 and 2. M. S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1980. (AD 115 510 and AD 115 511)

17. Floyd, R. M. Doctoral Candidate (personal conversation). Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 10 July - 31 October 1983.

18. Freudenberg, J. S., D. P. Looze and J. B. Cruz. "Robustness Analysis Using Singular Value Sensitivities," International Journal of Control, 35 (1): 95-116 (January 1982).

19. Gilbert, E. G. "Conditions for Minimizing the Norm Sensitivity of Characteristic Roots," Conference on Information Sciences and Systems, Baltimore, Maryland. (March 1983).

20. Harvey, C. H. and G. Stein. "Quadratic Weights for Asymptotic Regulator Properties," IEEE Transactions on Automatic Control, AC-23 (3): 378-387 (June 1978).

21. Horowitz, I. M. Synthesis of Feedback Systems. New York: Academic Press, 1963.

22. Howey, J. M. Robust Flight Controllers. M. S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1983.

23. Kleinman, D. L. "A Description of Computer Programs Useful in Linear Systems Studies," Tec. Rep. TR-75-4. University of Connecticut, Storrs, Connecticut, October 1975.

24. Klema, V. C. and A. J. Laub. "The Singular Value Decomposition: Its Computation and Some Applications," IEEE Transactions on Automatic Control, AC-25 (2): 164-176 (April 1980)

25. Kreindler, E. and D. Rothschild. "Model-Following in Linear-Quadratic Optimization," AIAA Journal (7): 835-842 (July 1976).

26. Kwakernaak, H. and R. Sivan. Linear Optimal Control Systems. New York: John Wiley and Sons, Inc., 1972.

27. Lehtomaki, N. A., D. Castanon, B. Levy, G. Stein, N. R. Sandell, Jr., and M. Athans. "Robustness Tests Utilizing the Structure of Modelling Error," Proceedings of the 20th IEEE Conference on Decision and Control, San Diego, California. 1173-1190 (December 1981).

28. Lloyd, E. D. Robust Control Systems. M. S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1981. (AD 115 478)

29. Maybeck, P. S. Stochastic Models, Estimation, and Control, Volume 1. New York: Academic Press, 1979.

30. Maybeck, P. S. Stochastic Models, Estimation, and Control, Volume 3. New York: Academic Press, 1982.

31. Maybeck, P. S., R. M. Floyd and A. Moseley. "Synthesis and Performance Evaluation Tools for CGT/PI Advanced Digital Flight Control Systems," IEEE 1983 National Aerospace and Electronics Conference (NAECON 1983), Dayton, Ohio. 1259-1266 (May 1983).

32. McMillian, J. P. Command Generator Tracker Synthesis Methods Using an LQG-Derived Proportional Plus Integral Controller Based on the Integral of the Regulation Error. M. S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1983.

33. Moore, B. C. "On the Flexibility Offered by State Feedback in Multivariable Systems Beyond Closed Loop Eigenvalue Assignment," IEEE Transactions on Automatic Control, AC-21 (5): 689-692 (October 1976).

34. Moseley, A. Design of Advanced Digital Flight Control Systems Via Command Generator Tracker (CGT) Synthesis Techniques, Volumes 1 and 2. M. S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1982.

35. Mukhopadhyay, V. "Application of Matrix Singular Value Properties for Evaluating Gain and Phase Margins of Multiloop Systems," AIAA Guidance and Control Conference, San Diego, California. 420-428 (August 1982).

36. Musick, S. H. "SOFE: A Generalized Digital Simulation for Optimal Filter Evaluation; User's Manual," T.R. AFWAL-TR-80-1108, Air Force Wright Aeronautical Laboratories, Avionics Laboratory, Wright-Patterson Air Force Base, Ohio, 1980.

37. Safonov, M. G., A. J. Laub and G. L. Hartmann. "Feedback Properties of Multivariable Systems: The Role and Use of the Return Difference Matrix," *IEEE Transactions on Automatic Control*, *AC-26* (1): 47-65 (February 1981).

38. Safonov, M. G. "Stability Margins of Diagonally Perturbed Multivariable Feedback Systems," *IEE Proceedings*, *Vol 129* (D,6): 251-256 (November 1982).

39. Safonov, M. G. and M. Athans. "Gain and Phase Margins for Multiloop LQG Regulators," *IEEE Transactions on Automatic Control*, *AC-22* (2): 173-179 (April 1977).

40. Safonov, M. G. and B. S. Chen. "Multivariable Stability-Margin Optimisation with Decoupling and Output Regulation," *IEE Proceedings*, *Vol 129* (D,6): 276-282 (November 1982).

41. Sandell, N. R. "Robust Stability of Systems with Application to Singular Perturbations," *Automatica*, *15* (4): 467-470 (July 1979).

42. Shampine, L. F. and M. K. Gordon. *Computer Solution of Ordinary Differential Equations*. San Francisco: W. H. Freeman and Company, 1975.

43. Stein G. and N. R. Sandell. Notes for Subject 6.291. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1979.

A. LQG Regulator Synthesis

A.1 Introduction

This thesis frequently refers to inner-loop controller designs based on linear system models, quadratic costs and Gaussian disturbances (the LQG assumptions). The PI regulator used in the CGT/PI/KF controller is a sampled-data optimal controller based on these LQG assumptions. Chapter II briefly outlines the overall structure of the CGT/PI/KF controller as well as the concept of the alteration of closed-loop system characteristics by means of implicit model-following. In that chapter, it is assumed that the reader understands the fundamental synthesis techniques involved in LQG designs, for both simple regulators and PI regulators.

This appendix is a very brief review of the development of those LQG synthesis techniques, intended to introduce concepts and notation used elsewhere in the thesis and in the design software that was employed in this study. Emphasis is on sampled-data controllers that are based on models representing actual discrete-time systems, or on equivalent discrete-time models for continuous-time systems. For a more complete and general development, refer to [30]; the specific formulation upon which the designs of this study are based is included in [16,34].

A.2 A Simple Constant-Gain LQG Regulator

Consider a linear, time-invariant system which can be described by the vector state equation

$$\underline{x}(t_{i+1}) = \underline{\Phi x}(t_i) + \underline{B}_d\underline{u}(t_i) + \underline{G}_d\underline{w}_d(t_i) \tag{A-1}$$

The system states at discrete sample times are represented by the vector $\underline{x}(t_i)$, and $\underline{u}(t_i)$ is the control applied at time t_i, and held constant until time t_{i+1}. If the model represents an actual discrete process, then $\underline{w}_d(t_i)$ is a zero-mean, discrete-time white Gaussian driving noise of covariance $\underline{Q}_d$ applied to the states through the distribution matrix $\underline{G}_d$. Often the model is an equivalent discrete-time description of an underlying continuous-time process [30]; in such a case, the equivalent discrete-time control matrix and driving noises can be developed from the related continuous-time quantities as

$$\underline{B}_d = \int_{t_i}^{t_{i+1}} \underline{\Phi}(t_{i+1},\tau)\underline{B}d\tau \tag{A-2a}$$

$$\underline{G}_d = \underline{I} \tag{A-2b}$$

and

$$\underline{Q}_d = \int_{t_i}^{t_{i+1}} \underline{\Phi}(t_{i+1},\tau)\underline{GQG}^T\underline{\Phi}^T(t_{i+1},\tau)d\tau \tag{A-2c}$$

where $\underline{\Phi}(t_{i+1},\tau)$ is the state transition matrix associated with $\underline{A}$ in the continuous-time model

$$\dot{\underline{x}}(t) = \underline{Ax}(t) + \underline{Bu}(t) + \underline{Gw}(t) \tag{A-3a}$$

and $\underline{Q}$ is the strength of the white Gaussian noise $\underline{w}(t)$:

$$E\{\underline{w}(t)\underline{w}^T(t+\tau)\} = \underline{Q}\delta(\tau) \tag{A-3b}$$

Note that if $\underline{A}$, $\underline{B}$, $\underline{G}$, and $\underline{Q}$ are constant and the sample period is fixed, then the matrices in (A-2) need only be evaluated once for all sample periods. Also, the $\underline{\Phi}$ of (A-1) is just $\underline{\Phi}(t_{i+1},t_i)$ of (A-2) and, again, it is the same for all sample periods.

The system outputs are a linear combination of the system states

$$\underline{y}(t_i) = \underline{Cx}(t_i) \qquad \text{(A-4)}$$

Equation (A-4) may be generalized to include direct feedthrough of the control inputs to the system outputs, but doing so will change neither the most general form of the quadratic cost function nor the form of the controller. Such a generalization is therefore unnecessary at this point in the development of a simple regulator [30]. Feedback control is based on noise corrupted sampled-data measurements of the system states in the form

$$\underline{z}(t_i) = \underline{Hx}(t_i) + \underline{v}(t_i) \qquad \text{(A-5)}$$

where $\underline{v}(t_i)$ is a zero-mean, discrete-time white Gaussian measurement noise of covariance $\underline{R}$, which is independent of $\underline{w}_d(t_i)$.

Under the LQG assumptions, the optimal controller for the system just described consists of an optimal (LQ) deterministic controller cascaded with an optimal linear Kalman filter, and the design of the controller and the filter can be conducted separately; this is an important property known as certainty equivalence [30]. Thus, the controller can be designed based on a simpler deterministic model consisting of (A-4) and

$$\underline{x}(t_{i+1}) = \underline{\Phi x}(t_i) + \underline{B}_d\underline{u}(t_i) \qquad \text{(A-6)}$$

The result is a linear-quadratic-state-feedback (LQSF) controller; in the final implementation, the state values used for feedback may be replaced by Kalman filter estimates to produce an LQG regulator.

The objective of the controller is to regulate all of the system outputs to zero, without expending excessive amounts of control energy. The determination of a control law, of the form

$$\underline{u}(t_i) = -\underline{G}_c(t_i)\underline{x}(t_i) \tag{A-7}$$

which meets these criteria for N sample periods can be accomplished by minimization of a quadratic cost function (performance index) of the form

$$J = \sum_{i=0}^{N} 1/2[\underline{y}^T(t_i)\underline{Y}_S(t_i)\underline{y}(t_i) + \underline{u}^T(t_i)\underline{U}_S(t_i)\underline{u}(t_i)] + 1/2[\underline{y}^T(t_{N+1})\underline{Y}_f\underline{y}(t_{N+1})] \tag{A-8}$$

where $\underline{Y}_S(t_i)$, $\underline{Y}_f$ and $\underline{U}_S(t_i)$ are symmetric, positive definite quadratic weighting matrices assigned by the designer in order to achieve the design objectives [30]. In many cases (as with the CGT/PI controller) the terminal transient can be ignored and time-invariant weighting matrices used to determine a more easily implemented constant-gain steady-state control law

$$\underline{u}(t_i) = -\underline{G_c x}(t_i) \tag{A-9}$$

through the minimization of

$$J = \sum_{i=0}^{\infty} 1/2[\underline{y}^T(t_i)\underline{Y_S y}(t_i) + \underline{u}^T(t_i)\underline{U_S u}(t_i)] \tag{A-10}$$

or

$$J = \sum_{i=0}^{\infty} 1/2[\underline{x}^T(t_i)\underline{X}_S\underline{x}(t_i) + \underline{u}^T(t_i)\underline{U}_S\underline{u}(t_i)] \tag{A-11}$$

where, in view of (A-4),

$$\underline{X}_S = \underline{C}^T\underline{Y}_S\underline{C} \tag{A-12}$$

Although $\underline{Y}_S$ is naturally positive definite, (A-12) reveals that $\underline{X}_S$ may well be only positive semi-definite.

In the case of a controller based on an equivalent discrete-time model of an underlying continuous-time system, it is often convenient to specify the cost initially as a continuous-time function, such as those used for the example model-following development of Section 2.2. This form of cost can then be discretized [16,30] to produce a discrete-time cost similar to that in (A-11), but generalized to include cross-weighting terms between the system states and controls. Such a form generally arises when the discrete-time quadratic cost function is developed through the discretization of a continuous-time cost function, even if the continuous-time function contains no cross-weightings [30]. It would also occur if (A-4) were modified to admit direct feedthrough. This generalized cost function is

$$J = \sum_{i=0}^{\infty} 1/2 \begin{bmatrix} \underline{x}(t_i) \\ \underline{u}(t_i) \end{bmatrix}^T \begin{bmatrix} \underline{X}_S & \underline{S} \\ \underline{S}^T & \underline{U}_S \end{bmatrix} \begin{bmatrix} \underline{x}(t_i) \\ \underline{u}(t_i) \end{bmatrix} \tag{A-13}$$

where $\underline{S}$ is the cross-weighting matrix; $\underline{U}_S$ is still assumed positive definite and the composite matrix in (A-13) is assumed positive semi-definite. Based on such a generalized quadratic cost function, the determination of the constant-gain, steady-state control law requires only the solution of the algebraic matrix Riccati equation [30]

$$\underline{K}_c = \underline{X}_S + \underline{\Phi}^T \underline{K}_c \underline{\Phi} - [\underline{B}_d^T \underline{K}_c \underline{\Phi} + \underline{S}^T]^T \underline{G}_c \tag{A-14}$$

where

$$\underline{G}_c = [\underline{U}_S + \underline{B}_d^T \underline{K}_c \underline{B}_d]^{-1} [\underline{B}_d^T \underline{K}_c \underline{\Phi} + \underline{S}^T] \tag{A-15}$$

Software is available for the efficient solution of such equations [23]. In practice, the use of standard matrix Riccati equation solving routines often requires that the weighting matrix of (A-13) be transformed to a form with no cross-weighting term. Once the solution for the transformed system is found, an inverse transformation yields the optimal steady-state control law for the original system [16,23,26,30].

A.3 A Constant-Gain PI Regulator

In many applications, the simple regulator developed in the preceding section is not suitable, because a controller is needed that will regulate the system outputs to non-zero values. Such is the case of the CGT/PI wherein the objective is to cause the system outputs to match the model outputs represented by $\underline{y}_m(t_i)$ of (II-18). An additional objective is to cause the error quantity

$$\underline{e}(t_i) = \underline{y}(t_i) - \underline{y}_m(t_i) \tag{II-19}$$

to be zero in the steady state despite any unmodelled constant disturbances. A controller which can satisfy these criteria is said to have the "type-1 property" that can be achieved with a

Proportional-plus-Integral (PI) regulator [8]. Such a regulator can be designed by applying the LQ methodology introduced in the preceding section. This methodology provides a systematic synthesis procedure for a multiple-input, multiple-output controller which ensures stability under design conditions, and which includes the appropriate crossfeeds based on the system and cost descriptions. By iteratively adjusting the weights used in the quadratic cost function, the designer can trade off state and control amplitudes so as to achieve performance specifications [30]. The development of a full-state, constant-gain, LQ perturbation regulator with proportional-plus-integral characteristics follows; this deterministic optimal controller can be cascaded with a Kalman filter, based on certainty equivalence, to produce the LQG PI regulator [30].

The system of (A-6) is unchanged except for the addition of a constant disturbance, d, of unknown magnitude

$$\underline{x}(t_{i+1}) = \underline{\Phi}\underline{x}(t_i) + \underline{B}_d\underline{u}(t_i) + \underline{d} \tag{A-16}$$

and, in order to formulate the most general quadratic cost function, direct feedthrough of the inputs to the outputs is allowed:

$$\underline{y}(t_i) = \underline{C}\underline{x}(t_i) + \underline{D}_y\underline{u}(t_i) \tag{A-17}$$

If $\underline{y}_m(t_i)$ changes slowly in comparison to the response time of the system, then the development can be made based on a constant $\underline{y}_m$ [30]. The nominal state values, $\underline{x}_o$, and the nominal control, $\underline{u}_o$, can be determined for the case where $\underline{d}$ is zero, or

$$\underline{x}_o = \underline{\Phi}\underline{x}_o + \underline{B}_d\underline{u}_o \tag{A-18}$$

$$\underline{y}_m = \underline{C}\underline{x}_o + \underline{D}_y\underline{u}_o \tag{A-19}$$

$$\begin{bmatrix} \underline{\Phi} - \underline{I} & \underline{B}_d \\ \underline{C} & \underline{D}_y \end{bmatrix} \begin{bmatrix} \underline{x}_o \\ \underline{u}_o \end{bmatrix} = \begin{bmatrix} \underline{0} \\ \underline{y}_m \end{bmatrix} \tag{A-20}$$

$$\begin{bmatrix} \underline{\Phi} - \underline{I} & \underline{B}_d \\ \underline{C} & \underline{D}_y \end{bmatrix}^{-1} = \begin{bmatrix} \underline{\Pi}_{11} & \underline{\Pi}_{12} \\ \underline{\Pi}_{21} & \underline{\Pi}_{22} \end{bmatrix} \tag{A-21}$$

$$\underline{x}_o = \underline{\Pi}_{12}\underline{y}_m \tag{A-22}$$

$$\underline{u}_o = \underline{\Pi}_{22}\underline{y}_m \tag{A-23}$$

The case where $\underline{d}$ is non-zero requires the definition of perturbation variables that reflect deviations from the nominal conditions

$$\delta\underline{x}(t_i) \triangleq \underline{x}(t_i) - \underline{x}_o = \underline{x}(t_i) - \underline{\Pi}_{12}\underline{y}_m \tag{A-24}$$

$$\delta\underline{u}(t_i) \triangleq \underline{u}(t_i) - \underline{u}_o = \underline{u}(t_i) - \underline{\Pi}_{22}\underline{y}_m \tag{A-25}$$

$$\delta\underline{y}(t_i) \triangleq \underline{y}(t_i) - \underline{y}_m \tag{A-26}$$

and

$$\delta\underline{x}(t_{i+1}) = \underline{\Phi}\delta\underline{x}(t_i) + \underline{B}_d\delta\underline{u}(t_i) \tag{A-27}$$

$$\delta\underline{y}(t_i) = \underline{C}\delta\underline{x}(t_i) + \underline{D}_y\delta\underline{u}(t_i) \tag{A-28}$$

Since

$$\delta\underline{u}(t_{i+1}) - \delta\underline{u}(t_i) = [\underline{u}(t_{i+1}) - \underline{u}_o] - [\underline{u}(t_i) - \underline{u}_o] \qquad \text{(A-29)}$$

$$\delta\underline{u}(t_{i+1}) = \delta\underline{u}(t_i) + [\underline{u}(t_{i+1}) - \underline{u}(t_i)]$$

$$= \delta\underline{u}(t_i) + \Delta\underline{u}(t_i) \qquad \text{(A-30)}$$

where $\Delta\underline{u}(t_i)$ is a "pseudorate" that provides the control input to a perturbation regulator for the augmented system. Thus, the augmented perturbation state equation is

$$\begin{bmatrix} \delta\underline{x}(t_{i+1}) \\ \delta\underline{u}(t_{i+1}) \end{bmatrix} = \begin{bmatrix} \underline{\Phi} & \underline{B}_d \\ \underline{0} & \underline{I} \end{bmatrix} \begin{bmatrix} \delta\underline{x}(t_i) \\ \delta\underline{u}(t_i) \end{bmatrix} + \begin{bmatrix} \underline{0} \\ \underline{I} \end{bmatrix} \Delta\underline{u}(t_i) \qquad \text{(A-31)}$$

and the appropriate discrete time cost function to be minimized in defining a constant-gain control law is [30]

$$J = \sum_{i=0}^{\infty} 1/2 \begin{bmatrix} \delta\underline{x}(t_i) \\ \delta\underline{u}(t_i) \\ \Delta\underline{u}(t_i) \end{bmatrix}^T \begin{bmatrix} \underline{X}_{11} & \underline{X}_{12} & \underline{S}_1 \\ \underline{X}_{12}^T & \underline{X}_{22} & \underline{S}_2 \\ \underline{S}_1^T & \underline{S}_2^T & \underline{U} \end{bmatrix} \begin{bmatrix} \delta\underline{x}(t_i) \\ \delta\underline{u}(t_i) \\ \Delta\underline{u}(t_i) \end{bmatrix} \qquad \text{(A-33)}$$

As in case of the simple regulator, the required gains are found by solution of an algebraic Riccati equation, producing an incremental control law [30]:

$$\delta\underline{u}(t_{i+1}) = \delta\underline{u}(t_i) - \underline{G}_{c1}\delta\underline{x}(t_i) - \underline{G}_{c2}\delta\underline{u}(t_i) \qquad \text{(A-34)}$$

In order to achieve the desired "type-1 property" a signal proportional to the pseudointegral of the regulation error of (II-19) must be added. A suitable form for the resulting control law would be

[30]

$$\underline{u}(t_i) = \underline{u}(t_{i-1}) - \underline{K}_x[\underline{x}(t_i) - \underline{x}(t_{i-1})] + \underline{K}_z[\underline{y}_m - \underline{y}(t_{i-1})] \tag{A-35}$$

It can be shown [30] that both (A-34) and (A-35) are satisfied if

$$\underline{K}_x = \underline{G}_{c1}\underline{\Pi}_{11} + \underline{G}_{c2}\underline{\Pi}_{21} \tag{A-36}$$

$$\underline{K}_z = \underline{G}_{c1}\underline{\Pi}_{12} + \underline{G}_{c2}\underline{\Pi}_{22} \tag{A-37}$$

If $\underline{y}_m$ is permitted to change slowly, so that the system essentially reaches steady state before each subsequent change, the final PI regulator control law becomes [30]

$$\underline{u}(t_i) = \underline{u}(t_{i-1}) - \underline{K}_x[\underline{x}(t_i) - \underline{x}(t_{i-1})] + \underline{K}_z[\underline{y}_m(t_i) - \underline{y}(t_{i-1})] \tag{A-38}$$

where it is appropriate that the time arguments of the last term do not match [30].

A.4 Quadratic Cost and Implicit Model-Following for CGTPIV

The design software used in this study implements a PI regulator, as developed in the preceding section, as the inner-loop controller required to provide stability and regulation for the overall CGT/PI. The software requires the designer to establish continuous-time models for the system to be controlled; equivalent discrete-time models are developed internally by the program [16]. Since the designer works with a continuous-time model, the software also requires the input of quadratic weights for a continuous-time quadratic cost function. In the "standard" PI regulator of the original CGT/PI/KF design software [16], which is still an option of the current version [34], implicit model-following is not used; the continuous-time cost is specified in terms of a $\underline{Y}$ matrix which weights output deviations, a $\underline{U}_M$ matrix which weights the control magnitudes, and a $\underline{U}_R$ matrix which weights control rates. The continuous-time cost function is

$$J = 1/2\int_0^\infty [\delta\underline{y}^T(t)\underline{Y}\delta\underline{y}(t) + \delta\underline{u}^T(t)\underline{U}_M\delta\underline{u}(t) + \Delta\underline{u}^T(t)\underline{U}_R\Delta\underline{u}(t)]dt \tag{A-39}$$

The $\underline{Y}$ and $\underline{U}_M$ matrices are then combined with the output and feedthrough matrices to develop an $\underline{X}$ matrix:

$$\underline{X} = \begin{bmatrix} \underline{X}_{c11} & \underline{X}_{c12} \\ \underline{X}_{c12}^T & \underline{X}_{c22} \end{bmatrix} \tag{A-40}$$

where

$$\underline{X}_{c11} \triangleq \underline{C}^T\underline{YC} \tag{A-41a}$$

$$\underline{X}_{c22} \triangleq \underline{U}_M + \underline{D}_y^T\underline{YD}_y \tag{A-41b}$$

$$\underline{X}_{c12} \triangleq \underline{C}^T \underline{Y} \underline{D}_y \tag{A-41c}$$

The continuous-time cost function therefore becomes

$$J = 1/2 \int_0^\infty \begin{bmatrix} \delta\underline{x}(t) \\ \delta\underline{u}(t) \\ \Delta\underline{u}(t) \end{bmatrix}^T \begin{bmatrix} \underline{X}_{c11} & \underline{X}_{c12} & \underline{0} \\ \underline{X}_{c12}^T & \underline{X}_{c22} & \underline{0} \\ \underline{0} & \underline{0} & \underline{U}_R \end{bmatrix} \begin{bmatrix} \delta\underline{x}(t) \\ \delta\underline{u}(t) \\ \Delta\underline{u}(t) \end{bmatrix} dt \tag{A-42}$$

The program then discretizes the continuous-time cost function to produce the discrete-time cost function of (A-33) [16]. This process also introduces the required discrete-time cross-weightings.

The developments so far in this Appendix have not addressed the incorporation of the implicit model-following concept that was used so extensively in this study. The example development in Section 2.2 demonstrates that a continuous-time quadratic cost function incorporating an implicit model can be easily formulated. If the objective of the controller is to achieve simple regulation of outputs, then the development of Section 2.2 can be applied to the sampled-data controller, as well; discretization [16,30] of (II-5b) would result in a cost function of the same form as (A-13), but with weighting matrices appropriate for an implicit model-following regulator.

The development of implicit model-following for the PI regulator is also easy to envision in the context of a continuous-time cost, such as (A-39). If the objective of the controller developed in the preceding section is changed from regulating the perturbation outputs, defined as

$$\delta\underline{y}(t) = \underline{y}(t) - \underline{y}_m \tag{A-26}$$

to zero, to that of forcing those outputs to mimic the dynamics of a model system, so that

$$\delta\dot{\underline{y}}(t) = \underline{A}_m \delta\underline{y}(t) \tag{II-4}$$

then the appropriate continuous-time cost function would become

$$\begin{aligned} J = 1/2\int_0^\infty \Big\{ & [\delta\dot{\underline{y}}(t) - \underline{A}_m\delta\underline{y}(t)]^T\underline{Q}_I[\delta\dot{\underline{y}}(t) - \underline{A}_m\delta\underline{y}(t)] \\ & + \delta\underline{u}^T(t)\underline{R}_I\delta\underline{u}(t) + \Delta\underline{u}^T(t)\underline{U}_R\Delta\underline{u}(t)\Big\}\, dt \end{aligned} \tag{A-43}$$

or

$$J = 1/2\int_0^\infty \begin{bmatrix} \delta\underline{x}(t) \\ \delta\underline{u}(t) \\ \Delta\underline{u}(t) \end{bmatrix}^T \begin{bmatrix} \hat{\underline{Q}}_I & \hat{\underline{S}}_I & \underline{0} \\ \hat{\underline{S}}_I^T & \hat{\underline{R}}_I & \underline{0} \\ \underline{0} & \underline{0} & \underline{U}_R \end{bmatrix} \begin{bmatrix} \delta\underline{x}(t) \\ \delta\underline{u}(t) \\ \Delta\underline{u}(t) \end{bmatrix} dt \tag{A-44}$$

where

$$\hat{\underline{Q}}_I = (\underline{CA} - \underline{A}_m\underline{C})^T\underline{Q}_I(\underline{CA} - \underline{A}_m\underline{C}) \tag{A-45a}$$

$$\hat{\underline{S}}_I = \underline{B}^T\underline{C}^T\underline{Q}_I(\underline{CA} - \underline{A}_m\underline{C}) \tag{A-45b}$$

$$\hat{\underline{R}}_I = \underline{R}_I + \underline{B}^T\underline{C}^T\underline{Q}_I\underline{CB} \tag{A-45c}$$

Note that (A-45) is identical in form to (II-6), but that (A-43) through (A-45) are presented in notation consistent with CGTPIV and the discussions of Chapters IV and V.

The design path in CGTPIV that permits the use of implicit model-following actually implements a "combined explicit-implicit" regulator, based on a continuous-time cost function that results from combining (A-39) and (A-43) [34]. The designer has the freedom to specify the implicit regulator command model as well as $\underline{Y}$, $\underline{U}_M$, $\underline{U}_R$, $\underline{Q}_I$, and $\underline{R}_I$. For the " all-implicit" designs in this study, $\underline{Y}$ and $\underline{U}_M$ were set to zero. For the general combined case, instead of using the $\underline{X}$ matrix of (A-40) in the cost, as in (A-42), a combined state weighting matrix, $\underline{X}_{IE}$, is developed in CGTPIV by adding the implicit state weightings to $\underline{X}$ prior to discretizing the cost:

$$\underline{X}_{IE} = \underline{X} + \begin{bmatrix} \hat{\underline{Q}}_I & \hat{\underline{S}}_I \\ \hat{\underline{S}}_I^T & \hat{\underline{R}}_I \end{bmatrix} \tag{A-46}$$

The combined continuous-time cost function is, therefore

$$J = 1/2 \int_0^\infty \begin{bmatrix} \delta\underline{x}(t) \\ \delta\underline{u}(t) \\ \Delta\underline{u}(t) \end{bmatrix}^T \begin{bmatrix} (\underline{X}_{c11} + \hat{\underline{Q}}_I) & (\underline{X}_{c12} + \hat{\underline{S}}_I) & \underline{0} \\ (\underline{X}_{c12}^T + \hat{\underline{S}}_I^T) & (\underline{X}_{c22} + \hat{\underline{R}}_I) & \underline{0} \\ \underline{0} & \underline{0} & \underline{U}_R \end{bmatrix} \begin{bmatrix} \delta\underline{x}(t) \\ \delta\underline{u}(t) \\ \Delta\underline{u}(t) \end{bmatrix} dt \tag{A-47}$$

A.5 Summary

This appendix reviewed the formulation of basic LQG optimal regulator design methods that were used in developing the software that produced the designs of this study. The purpose was to ensure an adequate framework upon which to build an understanding of implicit model-following, and to relate the notation and terminology used in Chapters IV and V, as well as that of the CGTPIV design software, to that theoretical framework. The form of the PI regulator control law of (A-35) and of the CGT/PI control law of (II-25) is independent of whether the PI regulator gains are found by minimizing the "standard" cost of (A-39), the "implicit" cost of (A-43) or a combination of the two. However, as seen in the results of this study, the incorporation of an implicit model in the design of the regulator has a substantial beneficial effect on the closed-loop characteristics of the resulting controller.

B. CGTSVD Software Description and Instructions

B.1 Introduction

CGTSVD is an interactive computer program, developed for use in this study, which calculates the minimum and maximum singular values of the loop and inverse return difference matrix functions for any CGT/PI/KF or CGT/PI controller. This appendix develops the equations employed, and discusses the structure and use of CGTSVD; the information provided is intended to enhance the reader's ability not only to understand and use the software, but to modify it easily, if necessary. A short sample execution of the program is included, as is a complete listing of the source code.

CGTSVD was intentionally developed without the use of the sophisticated programming practices which made CGTPIV so efficient in terms of storage requirements and execution time [16]. Due to the comparatively small scope of the computational task, it was obvious at the outset that such practices would not be required in order to achieve a load size compatible with the normal interactive memory limits (either 65,000 or 100,000 octal words of storage) established for the Aeronautical Systems Division's CYBER computer. On the other hand, a degree of structural complexity was foreseen due to the effort to make the program as general as possible, able to handle a variety of calculations for a wide range of problems. For those reasons, the simplest programming methods were selected, so as to produce a source program that, although relatively inefficient, should be easily interpreted by anyone with experience in FORTRAN programming. The program is written in ANSI standard FORTRAN 77, and the source code includes many comments which explain what each segment of code does and, in many cases, how it is done. The program executes interactively, and prompts are sufficient in number and detail to lead a novice user along the proper paths. Because the source code is simple and reasonably self-explanatory, the discussion of the structure and use of the program is brief.

B.2 Development of Loop Equations

This section develops the equations used in CGTSVD for calculating the loop gain matrix function of a closed-loop CGT/PI or CGT/PI/KF controller. The use of the loop gain and inverse return difference (a function of the loop gain) matrix functions in robustness analysis is discussed at length in Section 3.2.

The overall control law for the closed-loop CGT/PI controller is given in (II-25). Since the loop gain does not include the effects of any quantities outside of the closed feedback loop, the command model states and commanded control inputs of (II-25) need not be considered. Additionally, the disturbance states only become a part of the loop if a Kalman filter is included to provide disturbance state estimates to the CGT controller. Unless stated otherwise, all of the quantities in the following development are as defined previously in Section 2.3.

For the LQSF (CGT/PI) controller, calculation of the loop gain function is quite simple once the quantities outside the feedback loop have been stripped from the control law. The remaining portion of the control law relates the system inputs to the system states:

$$\underline{u}(t_i) - \underline{u}(t_{i-1}) = - \underline{K}_x[\underline{x}(t_i) - \underline{x}(t_{i-1})] - \underline{K}_z[\underline{C}\underline{x}(t_{i-1}) + \underline{D}_y\underline{u}(t_{i-1})] \tag{B-1}$$

Transferring this difference equation to the z-domain [8],

$$\underline{u}(z) - \underline{u}(z)/z = - \underline{K}_x[\underline{x}(z) - \underline{x}(z)/z] - \underline{K}_z[\underline{C}\underline{x}(z)/z + \underline{D}_y\underline{u}(z)/z] \tag{B-2}$$

The PI controller "gain" representing the ratio of system inputs to system states is, therefore,

$$\underline{K}(z) = [(z-1)\underline{I} + \underline{K}_z\underline{D}_y]^{-1}[(1-z)\underline{K}_x - \underline{K}_z\underline{C}] \tag{B-3}$$

The plant "gain" representing the ratio of system states to system inputs is the Z transform [8] of the product of a zero-order hold (a device used to change the discrete control input at time t_i into a continuous input signal, held constant over the sample period, which drives the continuous-time plant [8]) and the s-domain representation of the continuous-time plant transfer function. In the s-domain, the zero-order hold device is represented as

$$ZOH = [1 - e^{-sT}]/s \tag{B-4}$$

where T is the controller sampling interval. The plant transfer function is

$$\underline{\Phi}(s)\underline{B} = [s\underline{I} - \underline{A}]^{-1}\underline{B} \tag{B-5}$$

where $\underline{A}$ and $\underline{B}$ are, respectively, the continuous-time dynamics and control matrices that make up the "evaluation model." The combined zero-order hold and plant is therefore

$$\underline{G}(z) = \text{Z transform } \left\{ [1 - e^{-sT}]/s\ [s\underline{I} - \underline{A}]^{-1}\underline{B} \right\} \quad \text{(B-6)}$$

and the loop gain, with the loop cut at the control input, is

$$\underline{G}_L(z) = \underline{K}(z)\underline{G}(z) \quad \text{(B-7a)}$$

and with the loop cut at the output,

$$\underline{G}_L(z) = \underline{G}(z)\underline{K}(z) \quad \text{(B-7b)}$$

In the implementation in CGTSVD, the user may specify the use of an "evaluation model" consisting of either the design model or a truth model description of the plant, which may be of higher dimension than the design model. When the truth model option is used, a transformation matrix must also be supplied which premultiplies the $\underline{G}(z)$ function so as to reduce the truth model state vector to the same dimension as, and quantities equivalent to, the design model state vector, thus making it compatible with the control law. The actual calculation of the loop gain singular values is accomplished, for each frequency, in the frequency (ω) domain, using the relationships [8]

$$s = j\omega \quad \text{(B-8)}$$

and

$$z = e^{j\omega T} \quad \text{(B-9)}$$

The corresponding development for the CGT/PI/KF is more complicated due to the need to calculate a "gain" or transfer function for the Kalman filter as well as the inclusion of the disturbance state estimates in the control law. In the development that follows, quantities that are filter estimates are depicted using a "hat" (^) notation. The portion of the control law relating the control inputs to the state estimates is

$$\begin{aligned} \underline{u}(t_i) - \underline{u}(t_{i-1}) = & - \underline{K}_x[\hat{\underline{x}}(t_i) - \hat{\underline{x}}(t_{i-1})] \\ & - \underline{K}_z[\underline{C}\hat{\underline{x}}(t_{i-1}) + \underline{D}_y\underline{u}(t_{i-1})] \\ & + \underline{K}_{xn}[\hat{\underline{n}}_d(t_i) - \hat{\underline{n}}_d(t_{i-1})] \end{aligned} \tag{B-10}$$

Transforming this difference equation to the z-domain and collecting terms yields

$$[(z-1)\underline{I} + \underline{K}_z\underline{D}_y]\underline{u}(z) = [\underline{K}_x(1-z) - \underline{K}_z\underline{C}]\hat{\underline{x}}(z) + \underline{K}_{xn}[(z-1)\hat{\underline{n}}_d(z)] \tag{B-11}$$

Defining an augmented state vector

$$\hat{\underline{x}}_a(z) = \begin{bmatrix} \hat{\underline{x}}(z) \\ \hdashline \hat{\underline{n}}_d(z) \end{bmatrix} \tag{B-12}$$

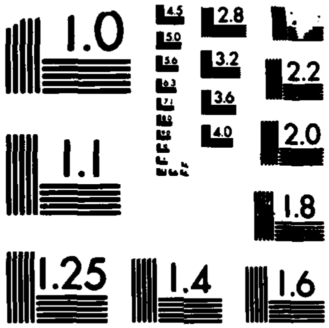
$$[(z-1)\underline{I} + \underline{K}_z\underline{D}_y]\underline{u}(z) = \left\{ [\underline{K}_x(1-z) - \underline{K}_z\underline{C}] \;\vdots\; [\underline{K}_{xn}(z-1)] \right\} \hat{\underline{x}}_a(z) \tag{B-13}$$

so that the ratio of control inputs to augmented state estimates is

$$\underline{K}_1(z) = [(z-1)\underline{I} + \underline{K}_z\underline{D}_y]^{-1} \left\{ [\underline{K}_x(1-z) - \underline{K}_z\underline{C}] \;\vdots\; [\underline{K}_{xn}(z-1)] \right\} \tag{B-14}$$

The overall "gain" or transfer function of the Kalman filter can be calculated from the propagation and update equations for the filter [29]. If a truth model state vector, $\underline{x}_t(t_i)$, is defined so as to include the "true" system and disturbance states, then the truth model

AD-A138 309 ROBUST MULTIVARIABLE CONTROLLER DESIGN VIA IMPLICIT 3/4
MODEL-FOLLOWING METHODS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W G MILLER
UNCLASSIFIED DEC 83 AFIT/GE/EE/83D-48 F/G 14/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

measurement vector is

$$\underline{z}_t(t_i) = \underline{H}_t\underline{x}_t(t_i) + \underline{v}_t(t_i) \tag{B-15}$$

and the design model measurement vector upon which the filter design is based is

$$\underline{z}(t\ ) = \underline{H}_a\underline{x}_a(t_i) + \underline{v}(t_i) \tag{B-16}$$

where

$$\underline{H}_a = [\underline{H} \;\vdots\; \underline{H}_n] \tag{II-27}$$

Using $\underline{K}_f$ to denote the Kalman filter gain,

$$\begin{aligned}\hat{\underline{x}}_a(t_i^+) &= \hat{\underline{x}}_a(t_i^-) + \underline{K}_f[\underline{z}_t(t_i) - \underline{H}_a\hat{\underline{x}}_a(t_i^-)] \\ &= [\underline{I} - \underline{K}_f\underline{H}_a]\hat{\underline{x}}_a(t_i^-) + \underline{K}_f[\underline{H}_t\underline{x}_t(t_i) + \underline{v}_t(t_i)] \end{aligned} \tag{B-17}$$

Since the measurement noises are outside of the loop, they can be dropped from the loop calculations, so that

$$\hat{\underline{x}}_a(t_i^+) = [\underline{I} - \underline{K}_f\underline{H}_a]\hat{\underline{x}}_a(t_i^-) + \underline{K}_f[\underline{H}_t\underline{x}_t(t_i)] \tag{B-18}$$

Also,

$$\hat{\underline{x}}_a(t_{i+1}^-) = \underline{\Phi}_a\hat{\underline{x}}_a(t_i^+) + \underline{B}_{da}\underline{u}(t_i) \tag{B-19a}$$

so

$$\hat{\underline{x}}_a(t_i^+) = \underline{\Phi}_a^{-1}\hat{\underline{x}}_a(t_{i+1}^-) - \underline{\Phi}_a^{-1}\underline{B}_{da}\underline{u}(t_i) \tag{B-19b}$$

where

$$\underline{\Phi}_a = \underline{\Phi}_a(t_{i+1},t_i) = \left[\begin{array}{c|c} \underline{\Phi} & \underline{E}_{xd} \\ \hline \underline{0} & \underline{\Phi}_n \end{array}\right] \tag{B-20a}$$

$$\underline{B}_a = \left[\begin{array}{c} \underline{B} \\ \hline \underline{0} \end{array}\right] \tag{B-20b}$$

and

$$\underline{B}_{da} = \int_{t_i}^{t_{i+1}} \underline{\Phi}_a(t_{i+1},\tau)\underline{B}_a d\tau \tag{B-20c}$$

Note that in (B-20c), $\underline{\Phi}_a(t_{i+1},\tau)$ is not the constant $\underline{\Phi}_a$ as in (B-20a), but is a variable in τ.

Z-transforming (B-18) and (B-19b) and using (B-14) to eliminate $\underline{u}(z)$ from the result gives a *transfer function* relating the augmented system state estimates to the truth model state realizations:

$$\underline{K}_2(z) = \left[z\underline{\Phi}_a^{-1} + \underline{K}_f\underline{H}_a - \underline{I} - \underline{\Phi}_a^{-1}\underline{B}_{da}\left\{[(z-1)\underline{I} + \underline{K}_z\underline{D}_y]^{-1} \cdot\left([\underline{K}_x(1-z) - \underline{K}_z\underline{C}] \,\vdots\, [\underline{K}_{xn}(z-1)]\right)\right\}\right]^{-1} \underline{K}_f\underline{H}_t \tag{B-21}$$

The existence of the disturbance state estimates within the loop is entirely accounted for in (B-14) and (B-21). The shaping filters for the disturbance states are outside the loop, so those states need not be represented in the plant evaluation dynamics model. The plant may again, therefore, be represented by an equation of the form of (B-6) where, in this case, the $\underline{A}$ and $\underline{B}$ matrices of the "evaluation model" represent the partitions of the model dynamics and control matrices associated with system states (i.e., excluding disturbances). For the "filter-in-the-loop" calculations, when the evaluation model state dimension exceeds that of the design model, the $\underline{H}_t$ matrix provides the necessary interface with the control law, and no additional transformation is required. Also, since the disturbances are not represented in the plant model, only the columns of the $\underline{H}_t$ matrix associated with system states need be included in the calculations (again, the disturbance states can be dropped). The overall loop gain, with the loop cut at the control input, is therefore

$$\underline{G}_L(z) = \underline{K}_1(z)\underline{K}_2(z)\underline{G}(z) \tag{B-22a}$$

and with the loop cut at the output,

$$\underline{G}_L(z) = \underline{G}(z)\underline{K}_1(z)\underline{K}_2(z) \tag{B-22b}$$

Once again, the actual calculations in CGTSVD are conducted in the ω-domain.

B.3 CGTSVD Program Structure

With the exception of the LINPACK [9] library routine CSVDC, which is used to find the singular values of complex matrices, the program CGTSVD is entirely self-contained. In the interest of simplicity, rather than efficiency, the majority of the calculations are conducted using complex arithmetic and with complex variable types. To allow the matrix manipulation subroutines to operate, as simply as possible, on partitions of larger arrays and on matrices of varying sizes, nearly all of the arrays in the main program and the subroutines are blocked with a common row dimension of 16. Despite the lack of effort to limit array storage requirements, the program, as listed in Section B.6, will handle problems with up to 16 augmented design model states (system states plus disturbance states, of which there may be 8), 16 truth model system states and 8 each control inputs and system outputs, while using 70,000 (octal) words of core memory.

The majority of the variable names used in the program (exclusive of work arrays) are reasonably well defined either by the comments in the source code, by the wording of the prompts that precede their definition by the user, or by their context. The flow of the program assumes that the design parameters will be read in from a previously established data file (line 780 of the source listing). If a data file is not used, the program branches to line 2600, where the parameters are entered from the terminal (this section of code is perhaps the best place to get familiar with the variable names used for the design parameters). Regardless of the way in which the parameters are entered, the user is given the option of changing any parameters via the "change menu" at line 1300 prior to each set of singular value calculations. As noted in the source listing, the logic followed in implementing changes is probably the most difficult to follow, since it requires jumping around through the data entry sequence to pick up all of the required information, without requiring the user to reenter data that does not change. Once the design parameters have been defined, the program branches to line 3950 to calculate singular values for the CGT/PI or to line 5110 for the CGT/PI/KF, as specified by the user. The program is terminated by a menu selection at line 1300.

In its current form, the program will calculate either the loop gain or inverse return difference function singular values, with the loop cut at either the control input or plant output. When the loop is cut at the plant output, the loop gain is generally rank deficient (the matrix is square, with dimension equal to either the design or evaluation model state dimension but with rank equal to the control input vector dimension). In such a case a pseudoinverse of the loop gain matrix is used in calculating the inverse return difference function. If an attempt is made to invert any rank deficient matrix under any other circumstances, the calculation is aborted, the user is advised, and the program returns to the "change menu" (line 1300).

The program produces printer plots, displayed at the user terminal and written to a plot file, of the logarithms of the absolute values of the minimum and maximum singular values for the selected function, versus radian frequency. Each plot includes 2 decades of frequency, or 37 points. To change the number of points per plot, it is only necessary to change the references to "37" in lines 390, 3950, 5200, 6510 and 6620 to the desired number and increase the size of the "PLTVEC" array in line 390 to 3 times the new number of points; for example, 55 (vice 37) would produce a plot covering 3 decades at the same frequency intervals. If the number of desired points exceeds 60, lines 11310 through 11330 should be deleted or modified, as they are designed to pad the line printer output to one full page per plot. The frequency interval may also be changed to affect the number of decades covered; for example, changing the 20.0 in lines 4960 and 6400 to 10.0 will halve the number of points plotted per decade. If the actual magnitude of the singular values is desired rather than the logarithm, the "LOG10" may be deleted in lines 4580, 4600, 4880, 4900, 6000, 6020, 6320 and 6340. The array dimensions may be altered to handle larger problems or reduce the load size, so long as the following are observed:

-- All of the arrays which are currently blocked with a row dimension of 16 in the main program and subroutines must always have the same common row dimension.

-- All of the two-dimensional "WORK" arrays should remain square.

B.4 Instructions for Using CGTSVD

Prior to executing CGTSVD, any data files established during previous sessions with the program should be attached, if they are to be used to define the design parameters. The program will allow repeated problem parameter redefinition through the use of multiple data files and keyboard inputs. Data files are distinguished by their file names, which are specified by the user. The data files for CGTSVD are not compatible with those of ODEACT or CGTPIV. Each time the problem is redefined, the user is given the opportunity to save the data to a new file; the program will not overwrite a previously established data file. The LINPACK subroutine library must also be attached and declared as a library prior to execution of CGTSVD.

Once program execution is begun, the user need only respond to the prompts for option selection and data input. The prompts are quite explicit as to what should be included with each entry. When a matrix is to be newly defined via keyboard input, the entire matrix must be entered, one row at a time, with the elements of the row separated by commas or spaces. When changes to a matrix are made, only the elements that are being redefined need be entered, in the format specified by the prompt.

The design and truth model dynamics and control matrices are entered in their continuous-time forms, as they are in CGTPIV. The continuous-time design model dynamics representation of the controlled system is [16]

$$\dot{\underline{x}}(t) = \underline{A}\underline{x}(t) + \underline{B}\underline{u}(t) + \underline{E}_x\underline{n}_d(t) + \underline{G}\underline{w}(t) \tag{B-21}$$

$$\dot{\underline{n}}_d(t) = \underline{A}_n\underline{n}_d(t) + \underline{G}_n\underline{w}_n(t) \tag{B-22}$$

so that when the program asks for the augmented design model dynamics matrix, the appropriate entry is of the form

$$\underline{A}_a = \left[\begin{array}{c|c} \underline{A} & \underline{E}_x \\ \hline \underline{0} & \underline{A}_n \end{array}\right] \tag{B-23}$$

and the design model control matrix is

$$\underline{B}_a = \left[\begin{array}{c} \underline{B} \\ \hline \underline{0} \end{array}\right] \tag{B-18b}$$

If a truth model is used as the "evaluation model" it is entered without any augmenting disturbance states. This means that the evaluation model dynamics, transformation, control and measurement matrices are to be stripped of all partitions relating to disturbance states. Assuming that previous prompts for model dimensions were answered correctly, the matrix entry prompts also provide reminders as to the correct dimensions for the models being entered. When the design model is used as the evaluation model, no evaluation model inputs are required.

For designs in which the number of control inputs exceeds the number of controlled outputs, the correct response to the "ENTER NUMBER OF OUTPUTS" prompt may be either the actual number of outputs or the number of control inputs. The entry that is used will define the number of rows of $\underline{C}$ and $\underline{D}_y$ (if used) as well as the number of columns of $\underline{K}_z$ that the input routines will expect. The only reason for choosing to enter the number of controls instead of the actual number of outputs would be so that the $\underline{C}$, $\underline{D}_y$, and $\underline{K}_z$ matrices would be the same as for the input/output of CGTPIV, where the number of outputs must equal the number of controls [16]. In such a case, the "extra" rows of $\underline{C}$ and $\underline{D}_y$ and columns of $\underline{K}_z$ will all be made up of zeroes, and do not affect the singular value calculations. When the actual number of outputs is entered in response to the prompt, these extra entries are not required. If the number of controlled outputs exceeds the number of control inputs (as was the case of the controllers in Section 5.5), the appropriate response to the "ENTER NUMBER OF CONTROLS" prompt is the actual number of control inputs available; in other words, no "dummy" controls, as used in CGTPIV, are required by CGTSVD. They will, in fact result in an abort of the calculations due to rank deficiency in the loop matrix.

To terminate the program, a "Y" response should be given to the "ANY CHANGES TO DATA" prompt; the "change menu" includes the option to exit the program. All of the singular value plots displayed at the terminal are automatically written to an output file, which is named by the user. This file may be saved for future reference or routed to a line printer to obtain a "hard" copy.

B.5 Sample Program Execution

The next few pages contain the terminal output from a short sample run of CGTSVD on the CYBER computer. The object code is contained in the file CGTSVD and the file AFTI5F contains data for the SR-B CGT/PI design defined in Section 5.2. The computer's prompts and outputs are displayed in upper case, and the non-numeric user responses in lower case type.

```
COMMAND- attach,afti5f

 PFN IS
 AFTI5F
 AT CY= ØØ1 SN=AFIT

COMMAND- attach,cgtsvd

 PFN IS
 CGTSVD
 AT CY= ØØ1 SN=AFIT

COMMAND- attach,linpack,id=library,sn=asd

 PFN IS
 LINPACK:
 AT CY= 999 SN=ASD

COMMAND- library,linpack

COMMAND- cgtsvd

ENTER NAME FOR PLOT OUTPUT FILE: plot1

DATA INPUT FROM TAPE? Y/N: y

ENTER NAME OF LOCAL FILE: afti5f

CURRENT STATUS: NO KALMAN FILTER
EVALUATION TRUTH MODEL = DESIGN MODEL
ANY CHANGES TO DATA? Y/N: n

ENTER SAMPLING TIME INTERVAL: .Ø2

ENTER STARTING FREQUENCY (RADIANS) AS A POWER OF 1Ø: Ø

1= SINGULAR VALUES OF LOOP MATRIX
2= SINGULAR VALUES OF INVERSE RETURN DIFFERENCE MATRIX
ENTER CHOICE #: 2

1= LOOP CUT AT CONTROL INPUT
2= LOOP CUT AT PLANT OUTPUT
ENTER CHOICE #: 1

+------------ENTER TITLE FOR PLOT---------------+
sample plot for sr-b design

1 = SINGLE PLOT, SINGLE SCALE
2 = TWO PLOTS -- ONE SINGLE SCALE, ONE DOUBLE SCALE
3 = SINGLE PLOT, DOUBLE SCALE
SPECIFY TYPE OF PLOT: 1
```

SAMPLE PLOT FOR SR-B DESIGN

```
  1.0000    2       1 +         +         +         +         +
  1.5000    + 2       +1        +         +         +         +
  2.0000    +  2      +  1      +         +         +         +
  2.5000    +    2    +   1     +         +         +         +
  3.0000    +      2  +     1   +         +         +         +
  3.5000    +       2 +      1  +         +         +         +
  4.0000    +         2       1 +         +         +         +
  4.5000    +         +2      1 +         +         +         +
  5.0000    +         + 2      1+         +         +         +
  5.5000    +: : : : :+: 2 : : :1: : : : :+: : : : :+: : : : :+
  6.0000    +         +   2     +1        +         +         +
  6.5000    +         +    2    +1        +         +         +
  7.0000    +         +     2   + 1       +         +         +
  7.5000    +         +     2   + 1       +         +         +
  8.0000    +         +      2  +  1      +         +         +
  8.5000    +         +       2 +  1      +         +         +
  9.0000    +         +        2+   1     +         +         +
  9.5000    +         +        2+   1     +         +         +
 10.0000    +         +         2   1     +         +         +
 15.0000    +: : : : :+: : : : :+: : 2 1 :+: : : : :+: : : : :+
 20.0000    +         +         +       21+         +         +
 25.0000    +         +         +        2+1        +         +
 30.0000    +         +         +         +21       +         +
 35.0000    +         +         +         + 21      +         +
 40.0000    +         +         +         +  21     +         +
 45.0000    +         +         +         +   21    +         +
 50.0000    +         +         +         +    21   +         +
 55.0000    +         +         +         +     21  +         +
 60.0000    +         +         +         +     2 1 +         +
 65.0000    +: : : : :+: : : : :+: : : : :+: : : 21:+: : : : :+
 70.0000    +         +         +         +      2 1+         +
 75.0000    +         +         +         +       2 1         +
 80.0000    +         +         +         +        21         +
 85.0000    +         +         +         +        2+1        +
 90.0000    +         +         +         +         21        +
 95.0000    +         +         +         +         2 1       +
100.0000    +         +         +         +         +21       +

  SCALE    -.100E+00 .500E+00  .110E+01  .170E+01  .230E+01  .290E+01
```

```
ANY CHANGES TO DATA? Y/N: y

1= EDIT MATRICES            5= ENTER ALL NEW DATA
2= ADD/DELETE EVAL MODEL    6= NEW DATA FROM TAPE
3= ADD/DELETE FILTER        7= NO CHANGES
4= SAVE DATA TO TAPE        8= EXIT PROGRAM
ENTER CHOICE: 8

      END  C6TSVD
      66600 MAXIMUM EXECUTION FL.
      0.678 CP SECONDS EXECUTION TIME.
COMMAND- route,plot1,dc=pr,st=csa,tid=af,fid=wgm
```

B.5 CGTSVD Source Listing

```
      PROGRAM CGTSVD
C
C**********************************************************************
C
C PROGRAM TO CALCULATE MINIMUM AND MAXIMUM SINGULAR VALUES OF
C THE LOOP AND INVERSE RETURN DIFFERENCE MATRIX FUNCTIONS FOR A
C CGT/PI/KF CONTROLLER.  USER SUPPLIES COMPLETE DESIGN AND TRUTH
C MODEL SPECIFICATIONS AS WELL AS CONTROLLER/FILTER GAINS.
C OPTIONS INCLUDE:
C     1.  LOOP/INVERSE RETURN DIFFERENCE FUNCTION SINGULAR VALUES
C     2.  LOOP CUT AT CONTROL INPUT OR PLANT OUTPUT
C     3.  ADDITION/REMOVAL OF TRUTH MODEL
C     4.  ADDITION/REMOVAL OF KALMAN FILTER
C
C DATE OF LAST REVISION: 16 OCT 83
C LIBRARIES USED:  LINPACK,ID=LIBRARY,SN=ASD
C
C**********************************************************************
C
      COMPLEX AWORK(16,16),BWORK(16,16),CWORK(16,16),DWORK(16,16)
      COMPLEX AMATE(16,16),KXNMAT(16,8),DYMAT(16,8),BMATE(16,8)
      COMPLEX PHIDSC(16,16),BDSC(16,8),SV(16),WORK(16),E(16)
      COMPLEX ZIDENT(16,16),ZOH,Z,S,ZLESS,U(16,16),V(16,16)
      COMPLEX AMAT(16,16),TDTMAT(16,16),BMAT(16,8),CMAT(16,16)
      COMPLEX HMAT(16,16),KFGAIN(16,16),KXMAT(16,16),KZMAT(16,8)
      COMPLEX HMATE(16,16)
      INTEGER IN,INDM,INDIS,IR,IP,IM,INAS,KFLAG,JFLAG
      INTEGER INEAS,LFLAG,MFLAG,INFO,INST,I,J,K
      INTEGER NFLAG,INFLAG,I1FLAG,I2FLAG
      REAL WRAD,TSAMP,DELW,WINC,RESULT(37,3),PLTVEC(120)
      REAL SCRACH(16,16)
      CHARACTER ANSW*1,TITLE*50,SAVE*6,DATA*6,PLOT*6
C
C**********************************************************************
C
C FLAG DEFINITIONS:
C I1FLAG = MENU CHOICE FOR CHANGES (0 MEANS NONE)
C I2FLAG = MENU CHOICE FOR EDITING MATRICES
C JFLAG = CHOICE OF LOOP FUNCTION (1) OR INVERSE RETURN DIFFERENCE (2)
C KFLAG = 1 IF KALMAN FILTER IN LOOP (0 IF LQSF)
C LFLAG = 1 IF LOOP CUT AT CONTROL INPUT (2 IF AT OUTPUT)
C MFLAG = NONZERO IF ATTEMPT MADE TO INVERT SINGULAR MATRIX
C NFLAG = 1 IF TRUTH MODEL USED (0 FOR DESIGN MODEL)
C INFLAG = MAX NUMBER OF BAD SINGULAR VALUES FROM CSVDC
C
C**********************************************************************
C
      KFLAG=0
      MFLAG=0
      NFLAG=0
C
C**********************************************************************
C
C PLOTS ARE AUTOMATICALLY SAVED TO TAPE (PLOT)
```

```
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
 5    PRINT*,'ENTER NAME FOR PLOT OUTPUT FILE: '
      READ(*,'(A)')PLOT
      OPEN(4,FILE=PLOT,STATUS='NEW',FORM='FORMATTED',ERR=5)
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C THIS SECTION ALLOWS PROBLEM DATA ENTRY FROM KEYBOARD OR TAPE (DATA)
C AND SAVING PROBLEM DATA TO TAPE (SAVE)
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
 10   PRINT*,'DATA INPUT FROM TAPE? Y/N: '
      READ(*,'(A)')ANSW
      IF (ANSW.NE.'Y'.AND.ANSW.NE.'N')THEN
         GO TO 10
      END IF
      IF (ANSW.EQ.'N')GO TO 100
C
C SECTION TO READ PROBLEM DATA FROM TAPE
C
 20   PRINT*,'ENTER NAME OF LOCAL FILE: '
      READ(*,'(A)')DATA
      OPEN(3,FILE=DATA,STATUS='OLD',FORM='UNFORMATTED',ERR=20)
      READ(3)KFLAG,NFLAG,INDM,INDIS,INAS,IR,IP
      READ(3)((AMAT(I,J),I=1,INDM),J=1,INDM)
      READ(3)((BMAT(I,J),I=1,INAS),J=1,IR)
      READ(3)((CMAT(I,J),I=1,IP),J=1,INAS)
      READ(3)((DYMAT(I,J),I=1,IP),J=1,IR)
      READ(3)((KXMAT(I,J),I=1,IR),J=1,INAS)
      READ(3)((KZMAT(I,J),I=1,IR),J=1,IP)
      IF(KFLAG.EQ.1)THEN
         READ(3)IM
         READ(3)((HMAT(I,J),I=1,IM),J=1,INDM)
         READ(3)((KFGAIN(I,J),I=1,INDM),J=1,IM)
         READ(3)((KXNMAT(I,J),I=1,IR),J=1,INDIS)
      END IF
 25   IF(NFLAG.EQ.1)THEN
         READ(3)INEAS
         READ(3)((AMATE(I,J),I=1,INEAS),J=1,INEAS)
         READ(3)((BMATE(I,J),I=1,INEAS),J=1,IR)
         READ(3)((TDTMAT(I,J),I=1,INAS),J=1,INEAS)
         IF(KFLAG.EQ.1) THEN
            READ(3)((HMATE(I,J),I=1,IM),J=1,INEAS)
         END IF
      ELSE
         INEAS=INAS
         CALL COPYMT(AMAT,AMATE,INAS,INAS)
         CALL COPYMT(BMAT,BMATE,INAS,IR)
         IF(KFLAG.EQ.1) THEN
            CALL COPYMT(HMAT,HMATE,IM,INAS)
         END IF
      END IF
      REWIND(3)
      CLOSE(3)
      IF(KFLAG.EQ.0) PRINT*,'CURRENT STATUS: NO KALMAN FILTER'
      IF(KFLAG.EQ.1) PRINT*,'CURRENT STATUS: KALMAN FILTER IN LOOP'
      IF(NFLAG.EQ.0) PRINT*,'EVALUATION TRUTH MODEL = DESIGN MODEL'
      IF(NFLAG.EQ.1) PRINT*,'TRUTH MODEL IN USE FOR EVALUATION'
C
C THIS SECTION ALLOWS CHANGES TO CURRENT PROBLEM DATA TO BE MADE:
C THE RIDE GETS A BIT BUMPY, SINCE WE WILL HAVE TO JUMP AROUND
C SOMEWHAT TO PICK UP ALL OF THE APPROPRIATE DATA FOR THE PROBLEM
C
   30 PRINT*,'ANY CHANGES TO DATA? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 30
      IF(ANSW.EQ.'N') GO TO 280
         PRINT*,'1= EDIT MATRICES              5= ENTER ALL NEW DATA'
         PRINT*,'2= ADD/DELETE EVAL MODEL      6= NEW DATA FROM TAPE'
         PRINT*,'3= ADD/DELETE FILTER          7= NO CHANGES'
         PRINT*,'4= SAVE DATA TO TAPE          8= EXIT PROGRAM'
   35    PRINT*,'ENTER CHOICE: '
         READ*,I1FLAG
         IF(I1FLAG.LT.1.OR.I1FLAG.GT.8) GO TO 35
   40    IF(I1FLAG.EQ.1)THEN
            PRINT*,'1= A (DESIGN)              8= KX'
            PRINT*,'2= A (EVALUATION)          9= KZ'
            PRINT*,'3= TRANSFORMATION          10= H (DESIGN)'
            PRINT*,'4= B (DESIGN)              11= H (EVALUATION)'
            PRINT*,'5= B (EVALUATION)          12= KFGAIN'
            PRINT*,'6= C                       13= KXN'
            PRINT*,'7= DY                      14= RETURN TO MENU'
   45       PRINT*,'ENTER CHOICE: '
            READ*,I2FLAG
            IF(I2FLAG.LT.1.OR.I2FLAG.GT.14) GO TO 45
            IF(I2FLAG.EQ.1) CALL EDIT(AMAT,INDM,INDM,SCRACH)
            IF(I2FLAG.EQ.2) THEN
               IF(NFLAG.EQ.1)THEN
                  CALL EDIT(AMATE,INEAS,INEAS,SCRACH)
               ELSE
                  NFLAG=1
                  GO TO 130
               END IF
            END IF
            IF(I2FLAG.EQ.3) THEN
               IF(NFLAG.EQ.1)THEN
                  CALL EDIT(TDTMAT,INAS,INEAS,SCRACH)
               ELSE
                  NFLAG=1
                  GO TO 130
               END IF
            END IF
            IF(I2FLAG.EQ.4) CALL EDIT(BMAT,INAS,IR,SCRACH)
            IF(I2FLAG.EQ.5) THEN
               IF(NFLAG.EQ.1)THEN
```

```
                  CALL EDIT(BMATE,INEAS,IR,SCRACH)
               ELSE
                  NFLAG=1
                  GO TO 130
               END IF
            END IF
            IF(I2FLAG.EQ.6) CALL EDIT(CMAT,IP,INAS,SCRACH)
            IF(I2FLAG.EQ.7) CALL EDIT(DYMAT,IP,IR,SCRACH)
            IF(I2FLAG.EQ.8) CALL EDIT(KXMAT,IR,INAS,SCRACH)
            IF(I2FLAG.EQ.9) CALL EDIT(KZMAT,IR,IP,SCRACH)
            IF(I2FLAG.GE.10.AND.I2FLAG.LE.13)THEN
               IF(KFLAG.EQ.0)THEN
                  KFLAG=1
                  GO TO 240
               ELSE
                  IF(I2FLAG.EQ.10) CALL EDIT(HMAT,IM,INDM,SCRACH)
                  IF(I2FLAG.EQ.11) CALL EDIT(HMATE,IM,INEAS,SCRACH)
                  IF(I2FLAG.EQ.12) CALL EDIT(KFGAIN,INDM,IM,SCRACH)
                  IF(I2FLAG.EQ.13) CALL EDIT(KXNMAT,IR,INDIS,SCRACH)
               END IF
            END IF
            IF(I2FLAG.EQ.14) GO TO 30
         GO TO 40
         END IF
         IF(I1FLAG.EQ.2) THEN
            IF(NFLAG.EQ.1) THEN
               NFLAG=0
               CALL COPYMT(AMAT,AMATE,INAS,INAS)
               CALL COPYMT(BMAT,BMATE,INAS,IR)
               IF(KFLAG.EQ.1) CALL COPYMT(HMAT,HMATE,IM,INAS)
               INEAS=INAS
               PRINT*,'EVALUATION MODEL DELETED'
            ELSE
               NFLAG=1
               GO TO 130
            END IF
         END IF
         IF(I1FLAG.EQ.3)THEN
            IF(KFLAG.EQ.1)THEN
               KFLAG=0
               PRINT*,'KALMAN FILTER DELETED'
            ELSE
               KFLAG=1
               GO TO 240
            END IF
         END IF
         IF(I1FLAG.EQ.4) GO TO 90
         IF(I1FLAG.EQ.5) GO TO 100
         IF(I1FLAG.EQ.6) GO TO 20
         IF(I1FLAG.EQ.7) GO TO 280
         IF(I1FLAG.EQ.8) GO TO 999
         GO TO 30
C
C SECTION TO SAVE PROBLEM DATA TO TAPE
```

```
C
   90  PRINT*,'ENTER NAME OF FILE FOR DATA SAVE: '
       READ(*,'(A)')SAVE
       OPEN(2,FILE=SAVE,STATUS='NEW',FORM='UNFORMATTED',ERR=90)
       WRITE(2)KFLAG,NFLAG,INDM,INDIS,INAS,IR,IP
       WRITE(2)((AMAT(I,J),I=1,INDM),J=1,INDM)
       WRITE(2)((BMAT(I,J),I=1,INAS),J=1,IR)
       WRITE(2)((CMAT(I,J),I=1,IP),J=1,INAS)
       WRITE(2)((DYMAT(I,J),I=1,IP),J=1,IR)
       WRITE(2)((KXMAT(I,J),I=1,IR),J=1,INAS)
       WRITE(2)((KZMAT(I,J),I=1,IR),J=1,IP)
       IF(KFLAG.EQ.1)THEN
          WRITE(2)IM
          WRITE(2)((HMAT(I,J),I=1,IM),J=1,INDM)
          WRITE(2)((KFGAIN(I,J),I=1,INDM),J=1,IM)
          WRITE(2)((KXNMAT(I,J),I=1,IR),J=1,INDIS)
       END IF
   95  IF(NFLAG.EQ.1)THEN
          WRITE(2)INEAS
          WRITE(2)((AMATE(I,J),I=1,INEAS),J=1,INEAS)
          WRITE(2)((BMATE(I,J),I=1,INEAS),J=1,IR)
          WRITE(2)((TDTMAT(I,J),I=1,INAS),J=1,INEAS)
          IF(KFLAG.EQ.1) THEN
             WRITE(2)((HMATE(I,J),I=1,IM),J=1,INEAS)
          END IF
       END IF
   98  ENDFILE(2)
       REWIND(2)
       CLOSE(2)
       GO TO 30
C
C NORMAL INPUTS FROM KEYBOARD
C NOTE: IN DYNAMICS MATRICES, ENTER DISTURBANCE STATES LAST
C
  100  I1FLAG=0
       PRINT*,'IS THERE A KALMAN FILTER IN THE LOOP? Y/N: '
       READ(*,'(A)')ANSW
       IF(ANSW.NE.'Y'.AND.ANSW.NE.'N')THEN
          GO TO 100
       ELSE IF(ANSW.EQ.'Y')THEN
          KFLAG=1
       ELSE
          KFLAG=0
       END IF
  110  PRINT*,'WILL THE DESIGN MODEL PLANT BE THE ONE USED TO'
       PRINT*,'EVALUATE THE SYSTEM? Y/N: '
       READ(*,'(A)')ANSW
       IF(ANSW.NE.'Y'.AND.ANSW.NE.'N')THEN
          GO TO 110
       ELSE IF (ANSW.EQ.'N')THEN
          NFLAG=1
       ELSE
          NFLAG=0
       END IF
```

```
      PRINT*,'ENTER NUMBER OF AUGMENTED DESIGN STATES: '
      READ*,INDM
      PRINT*,'ENTER NUMBER OF DISTURBANCE STATES IN DESIGN MODEL: '
      READ*,INDIS
      INAS=INDM-INDIS
      PRINT*,'ENTER NUMBER OF CONTROLS: '
      READ*,IR
 120  IF(KFLAG.EQ.1.OR.NFLAG.EQ.0)THEN
         PRINT*,'ENTER AUGMENTED DESIGN DYNAMICS MATRIX: '
         CALL CREAD(AMAT,INDM,INDM,SCRACH)
         PRINT*,'ENTER DESIGN CONTROL MATRIX: '
         CALL CREAD(BMAT,INAS,IR,SCRACH)
         IF(NFLAG.EQ.0)THEN
            CALL COPYMT(AMAT,AMATE,INAS,INAS)
            CALL COPYMT(BMAT,BMATE,INAS,IR)
            INEAS=INAS
         END IF
      END IF
 130  IF (NFLAG.EQ.1)THEN
         PRINT*,'ENTER NUMBER OF SYSTEM STATES IN EVALUATION'
         PRINT*,'DYNAMICS MATRIX: '
         READ*,INEAS
         PRINT*,'ENTER EVALUATION DYNAMICS MATRIX: '
         CALL CREAD(AMATE,INEAS,INEAS,SCRACH)
         PRINT*,'ENTER EVALUATION CONTROL MATRIX: '
         CALL CREAD(BMATE,INEAS,IR,SCRACH)
 140     PRINT*,'ENTER TRANSFORMATION MATRIX--EVALUATION TO'
         PRINT*,'DESIGN MODEL DIMENSIONS (SYSTEM STATES ONLY): '
         CALL CREAD(TDTMAT,INAS,INEAS,SCRACH)
      END IF
      IF(I1FLAG.EQ.2.AND.KFLAG.EQ.1) GO TO 250
      GO TO (40,30) I1FLAG
 150  PRINT*,'ENTER NUMBER OF OUTPUTS: '
      READ*,IP
      PRINT*,'ENTER OUTPUT MATRIX: '
      CALL CREAD(CMAT,IP,INAS,SCRACH)
 160  PRINT*,'IS THERE DIRECT INPUT FEEDTHROUGH ? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') THEN
         GO TO 160
      ELSE IF (ANSW.EQ.'Y')THEN
         PRINT*,'ENTER FEED THROUGH MATRIX: '
         CALL CREAD(DYMAT,IP,IR,SCRACH)
      ELSE
         DO 200 I=1,IP
            DO 200 J=1,IR
               DYMAT(I,J)=CMPLX(0.0,0.0)
 200     CONTINUE
      END IF
 220  PRINT*,'ENTER KX MATRIX: '
      CALL CREAD(KXMAT,IR,INAS,SCRACH)
 230  PRINT*,'ENTER KZ MATRIX: '
      CALL CREAD(KZMAT,IR,IP,SCRACH)
C
```

```
C DATA THAT FOLLOWS IS NOT NEEDED IF THE KALMAN FILTER IS NOT PRESENT
C
 240  IF(KFLAG.NE.1) GO TO 30
        PRINT*,'ENTER NUMBER OF MEASUREMENTS: '
        READ*,IM
        PRINT*,'ENTER H MATRIX FOR DESIGN MODEL: '
        CALL CREAD(HMAT,IM,INDM,SCRACH)
        PRINT*,'ENTER KALMAN FILTER GAINS: '
        CALL CREAD(KFGAIN,INDM,IM,SCRACH)
        PRINT*,'ENTER KXN MATRIX: '
        CALL CREAD(KXNMAT,IR,INDIS,SCRACH)
 250    IF(NFLAG.EQ.1) THEN
          PRINT*,'ENTER H MATRIX FOR EVALUATION MODEL: '
          CALL CREAD(HMATE,IM,INEAS,SCRACH)
        END IF
      GO TO (40,30,30) I1FLAG
      GO TO 30
C
C*********************************************************************
C
C END OF INPUT SECTION
C NOW BEGIN TO CALCULATE SINGULAR VALUES
C
C*********************************************************************
C
 280  PRINT*,'ENTER SAMPLING TIME INTERVAL: '
      READ*,TSAMP
      PRINT*,'ENTER STARTING FREQUENCY (RADIANS) AS A POWER OF 10: '
      READ*,IWST
      WRAD=10.0**IWST
      WRAD=WRAD+.000000001*ABS(WRAD)
      DO 300 I=1,16
        DO 290 J=1,16
          ZIDENT(I,J)=CMPLX(0.0,0.0)
 290    CONTINUE
        ZIDENT(I,I)=CMPLX(1.0,0.0)
 300  CONTINUE
      INFLAG=0
C
C USER MAY SPECIFY SINGULAR VALUE ANALYSIS OF THE LOOP MATRIX OR THE
C INVERSE RETURN DIFFERENCE MATRIX, WITH THE LOOP CUT AT THE INPUT
C OR THE OUTPUT.
C
 330  PRINT*,'1= SINGULAR VALUES OF LOOP MATRIX'
      PRINT*,'2= SINGULAR VALUES OF INVERSE RETURN DIFFERENCE MATRIX'
      PRINT*,'ENTER CHOICE #: '
      READ*,JFLAG
      IF (JFLAG.LT.1.OR.JFLAG.GT.2) GO TO 330
 350  PRINT*,'1= LOOP CUT AT CONTROL INPUT'
      PRINT*,'2= LOOP CUT AT PLANT OUTPUT'
      PRINT*,'ENTER CHOICE #: '
      READ*,LFLAG
      IF (LFLAG.LT.1.OR.LFLAG.GT.2) GO TO 350
      IF (KFLAG.EQ.1) GO TO 600
```

```
C
C**********************************************************************
C
C BRANCH TO CALCULATE SINGULAR VALUES WITH NO FILTER
C
C**********************************************************************
C
 400  DO 500 I=1,37
C
C CONTROLLER CALCULATIONS
C
          Z=EXP(CMPLX(0.0,TSAMP*WRAD))
          ZLESS=Z-1.0
          CALL CSMUL(ZLESS,ZIDENT,BWORK,IR,IR)
          CALL CMATML(KZMAT,DYMAT,AWORK,IR,IP,IR)
          CALL CMATAD(AWORK,BWORK,CWORK,IR,IR)
          CALL CMATIN(CWORK,CWORK,IR,DWORK,MFLAG)
          IF (MFLAG.NE.0)THEN
             PRINT*,'PROBLEM IS IN CONTROL LAW CALCULATION'
             PRINT*,'W = ',WRAD
             MFLAG=0
             GO TO 30
          END IF
 420      ZLESS=-ZLESS
          CALL CSMUL(ZLESS,KXMAT,AWORK,IR,INAS)
          CALL CMATML(KZMAT,CMAT,BWORK,IR,IP,INAS)
          CALL CMATSB(AWORK,BWORK,AWORK,IR,INAS)
          CALL CMATML(CWORK,AWORK,BWORK,IR,IR,INAS)
          CALL COPYMT(BWORK,CWORK,IR,INAS)
C
C PLANT CALCULATIONS AND CONVERSION TO DESIGN MODEL STATE DIMENSION
C
          ZOH=(1.0-EXP(CMPLX(0.0,-TSAMP*WRAD)))/CMPLX(0.0,WRAD)
          S=CMPLX(0.0,WRAD)
          CALL CSMUL(S,ZIDENT,AWORK,INEAS,INEAS)
          CALL CMATSB(AWORK,AMATE,AWORK,INEAS,INEAS)
          CALL CMATIN(AWORK,AWORK,INEAS,DWORK,MFLAG)
          IF (MFLAG.NE.0) THEN
             PRINT*,'PROBLEM IS IN PLANT MATRIX'
             PRINT*,'W = ',WRAD
             MFLAG=0
             GO TO 30
          END IF
 440      IF (INEAS.NE.INAS)THEN
             CALL CMATML(AWORK,BMATE,BWORK,INEAS,INEAS,IR)
             CALL CMATML(TDTMAT,BWORK,AWORK,INAS,INEAS,IR)
          ELSE
             CALL CMATML(AWORK,BMAT,BWORK,INAS,INAS,IR)
             CALL COPYMT(BWORK,AWORK,INAS,IR)
          END IF
          CALL CSMUL(ZOH,AWORK,AWORK,INAS,IR)
C
C CUT THE LOOP AND FORM THE FUNCTION TO BE PLOTTED
C
```

```
      IF(LFLAG.EQ.1)THEN
         CALL CMATML(CWORK,AWORK,BWORK,IR,INAS,IR)
         CALL CMATIN(BWORK,CWORK,IR,DWORK,MFLAG)
         IF (MFLAG.NE.0)THEN
            PRINT*,'PROBLEM IN LOOP MATRIX'
            PRINT*,'W = ',WRAD
            MFLAG=0
            GO TO 30
         END IF
 460     CALL CMATAD(CWORK,ZIDENT,AWORK,IR,IR)
         IF (JFLAG.EQ.2)THEN
            CALL CSVDC(AWORK,16,IR,IR,SV,E,U,16,V,16,WORK,0,INFO)
         ELSE
            CALL CSVDC(BWORK,16,IR,IR,SV,E,U,16,V,16,WORK,0,INFO)
         END IF
         RESULT(I,1)=WRAD
         RESULT(I,2)=LOG10(CABS(SV(INFO+1)))
         IF(INFO.GT.INFLAG) INFLAG=INFO
         RESULT(I,3)=LOG10(CABS(SV(IR)))
      ELSE
         CALL CMATML(AWORK,CWORK,BWORK,INAS,IR,INAS)
         CALL COPYMT(BWORK,DWORK,INAS,INAS)
         CALL CSVDC(BWORK,16,INAS,INAS,SV,E,U,16,V,16,WORK,22,INFO)
         CALL CXPOSE(U,AWORK,INAS,IR)
         DO 470 J=1,IR
            DO 465 K=1,IR
               CWORK(J,K)=CMPLX(0.0,0.0)
 465        CONTINUE
            CWORK(J,J)=SV(J)
 470     CONTINUE
         CALL CMATIN(CWORK,CWORK,IR,DWORK,MFLAG)
         IF(MFLAG.NE.0) THEN
            PRINT*,'PROBLEM IS IN PSEUDOINVERSE'
            PRINT*,'W = ',WRAD
            MFLAG=0
            GO TO 30
         END IF
         CALL CMATML(CWORK,AWORK,DWORK,IR,IR,INAS)
         CALL CMATML(V,DWORK,CWORK,INAS,IR,INAS)
 480     CALL CMATAD(CWORK,ZIDENT,AWORK,INAS,INAS)
         IF(JFLAG.EQ.2)THEN
            CALL CSVDC(AWORK,16,INAS,INAS,SV,E,U,16,V,16,WORK,0,INFO)
         ELSE
            CALL CSVDC(BWORK,16,INAS,INAS,SV,E,U,16,V,16,WORK,0,INFO)
         END IF
         RESULT(I,1)=WRAD
         RESULT(I,2)=LOG10(CABS(SV(INFO+1)))
         IF(INFO.GT.INFLAG) INFLAG=INFO
         RESULT(I,3)=LOG10(CABS(SV(IR)))
      END IF
C
C INCREMENT FREQUENCY AND REPEAT SINGULAR VALUE CALCULATION
C
      DELW=INT((LOG(WRAD))/LOG(10.0))
```

```
         WINC=(EXP(LOG(10.0)*DELW))*10.0/20.0
         IF(WRAD.GE.1.0)WRAD=WRAD+WINC
         IF(WRAD.LT.1.0)WRAD=WRAD+WINC/10.
  500 CONTINUE
      GO TO 900
C
C*******************************************************************
C
C BRANCH TO CALCULATE SINGULAR VALUES WITH KALMAN FILTER
C
C*******************************************************************
C
C
C CONTROLLER CALCULATIONS
C
  600 CALL CDSCRT(AMAT,INDM,TSAMP,PHIDSC,BDSC,30,AWORK,BWORK,CWORK)
      CALL CMATIN(PHIDSC,PHIDSC,INDM,DWORK,MFLAG)
      IF(MFLAG.NE.0)THEN
         PRINT*,'PROBLEM IS IN TRANSITION MATRIX'
         MFLAG=0
         GO TO 30
      END IF
  605 CALL CMATML(BDSC,BMAT,AWORK,INDM,INAS,IR)
      CALL COPYMT(AWORK,BDSC,INDM,IR)
  610 DO 700 I=1,37
         Z=EXP(CMPLX(0.0,TSAMP*WRAD))
         CALL CMATML(KZMAT,CMAT,AWORK,IR,IP,INAS)
         ZLESS=1.0-Z
         CALL CSMUL(ZLESS,KXMAT,BWORK,IR,INAS)
         CALL CMATSB(BWORK,AWORK,AWORK,IR,INAS)
         ZLESS=-ZLESS
         CALL CSMUL(ZLESS,KXNMAT,BWORK,IR,INDIS)
         DO 620 J=1,IR
            DO 615 K=1,INAS
               CWORK(J,K)=AWORK(J,K)
  615       CONTINUE
            DO 620 K=INAS+1,INDM
               CWORK(J,K)=BWORK(J,K-INAS)
  620    CONTINUE
         CALL CSMUL(ZLESS,ZIDENT,AWORK,IR,IR)
         CALL CMATML(KZMAT,DYMAT,BWORK,IR,IP,IR)
         CALL CMATAD(AWORK,BWORK,AWORK,IR,IR)
         CALL CMATIN(AWORK,BWORK,IR,DWORK,MFLAG)
         IF(MFLAG.NE.0)THEN
            PRINT*,'PROBLEM IS IN CONTROL LAW CALCULATIONS'
            PRINT*,'W = ',WRAD
            MFLAG=0
            GO TO 30
         END IF
  625    CALL CMATML(BWORK,CWORK,AWORK,IR,IR,INDM)
         CALL COPYMT(AWORK,CWORK,IR,INDM)
         CALL CMATML(BDSC,CWORK,AWORK,INDM,IR,INDM)
         CALL CMATML(PHIDSC,AWORK,BWORK,INDM,INDM,INDM)
         CALL CMATML(KFGAIN,HMAT,AWORK,INDM,IM,INDM)
      CALL CMATSB(AWORK,ZIDENT,AWORK,INDM,INDM)
      CALL CMATSB(AWORK,BWORK,BWORK,INDM,INDM)
      CALL CSMUL(Z,PHIDSC,AWORK,INDM,INDM)
      CALL CMATAD(AWORK,BWORK,BWORK,INDM,INDM)
      CALL CMATIN(BWORK,BWORK,INDM,DWORK,MFLAG)
      IF(MFLAG.NE.0)THEN
         PRINT*,'PROBLEM IN CONTROL LAW CALCULATION'
         PRINT*,'W = ',WRAD
         MFLAG=0
         GO TO 30
      END IF
 630  CALL CMATML(CWORK,BWORK,AWORK,IR,INDM,INDM)
      CALL CMATML(AWORK,KFGAIN,BWORK,IR,INDM,IM)
      CALL CMATML(BWORK,HMATE,AWORK,IR,IM,INEAS)
      ZOH=(1.0-EXP(CMPLX(0.0,-TSAMP*WRAD)))/CMPLX(0.0,WRAD)
      CALL CSMUL(ZOH,AWORK,AWORK,IR,INDM)
C
C PLANT CALCULATIONS
C
      S=CMPLX(0.0,WRAD)
      CALL CSMUL(S,ZIDENT,BWORK,INEAS,INEAS)
      CALL CMATSB(BWORK,AMATE,BWORK,INEAS,INEAS)
      CALL CMATIN(BWORK,BWORK,INEAS,DWORK,MFLAG)
      IF(MFLAG.NE.0)THEN
         PRINT*,'PROBLEM IN PLANT CALCULATION'
         PRINT*,'W = ',WRAD
         MFLAG=0
         GO TO 30
      END IF
      CALL CMATML(BWORK,BMATE,CWORK,INEAS,INEAS,IR)
      CALL COPYMT(CWORK,BWORK,INEAS,IR)
C
C CUT THE LOOP AND FORM THE FUNCTION TO BE PLOTTED
C
      IF(LFLAG.EQ.1) THEN
         CALL CMATML(AWORK,BWORK,CWORK,IR,INEAS,IR)
         CALL CMATIN(CWORK,AWORK,IR,DWORK,MFLAG)
         IF(MFLAG.NE.0)THEN
            PRINT*,'PROBLEM IN LOOP FUNCTION INVERSION'
            PRINT*,'W = ',WRAD
            MFLAG=0
            GO TO 30
         END IF
         CALL CMATAD(AWORK,ZIDENT,BWORK,IR,IR)
 640     IF(JFLAG.EQ.2) THEN
            CALL CSVDC(BWORK,16,IR,IR,SV,E,U,16,V,16,WORK,0,INFO)
         ELSE
            CALL CSVDC(CWORK,16,IR,IR,SV,E,U,16,V,16,WORK,0,INFO)
         END IF
         RESULT(I,1)=WRAD
         RESULT(I,2)=LOG10(CABS(SV(INFO+1)))
         IF(INFO.GT.INFLAG) INFLAG=INFO
         RESULT(I,3)=LOG10(CABS(SV(IR)))
      ELSE
```

```
            CALL CMATML(BWORK,AWORK,CWORK,INEAS,IR,INEAS)
            CALL COPYMT(CWORK,DWORK,INEAS,INEAS)
            CALL CSVDC(DWORK,16,INEAS,INEAS,SV,E,U,16,V,16,WORK,22,INFO)
            CALL CXPOSE(U,AWORK,INEAS,IR)
            DO 660 J=1,IR
               DO 650 K=1,IR
                  BWORK(J,K)=CMPLX(0.0,0.0)
 650           CONTINUE
               BWORK(J,J)=SV(J)
 660        CONTINUE
            CALL CMATIN(BWORK,BWORK,IR,DWORK,MFLAG)
            IF(MFLAG.NE.0) THEN
               PRINT*,'PROBLEM IS IN PSEUDOINVERSE'
               PRINT*,'W = ',WRAD
               MFLAG=0
               GO TO 30
            END IF
            CALL CMATML(BWORK,AWORK,DWORK,IR,IR,INEAS)
            CALL CMATML(V,DWORK,BWORK,INEAS,IR,INEAS)
 680        CALL CMATAD(BWORK,ZIDENT,AWORK,INEAS,INEAS)
            IF(JFLAG.EQ.2) THEN
               CALL CSVDC(AWORK,16,INEAS,INEAS,SV,E,U,16,
     1           V,16,WORK,0,INFO)
            ELSE
               CALL CSVDC(CWORK,16,INEAS,INEAS,SV,E,U,16,
     1           V,16,WORK,0,INFO)
            END IF
            RESULT(I,1)=WRAD
            RESULT(I,2)=LOG10(CABS(SV(INFO+1)))
            IF(INFO.GT.INFLAG) INFLAG=INFO
            RESULT(I,3)=LOG10(CABS(SV(IR)))
         END IF
C
C INCREMENT FREQUENCY AND REPEAT SINGULAR VALUE CALCULATION
C
         DELW=INT((LOG(WRAD))/LOG(10.0))
         WINC=(EXP(LOG(10.0)*DELW))*10.0/20.0
         IF(WRAD.GE.1.0) WRAD=WRAD+WINC
         IF(WRAD.LT.1.0) WRAD=WRAD+WINC/10.0
 700  CONTINUE
C
C*********************************************************************
C
C SET UP RESULTS AND CALL LINE PLOTTER
C
C*********************************************************************
C
 900  CALL SETPLT(RESULT,37,3,PLTVEC)
      PRINT*,'+------------ENTER TITLE FOR PLOT----------------+'
      PRINT*
      READ(*,'(A)')TITLE
      PRINT*,'1 = SINGLE PLOT, SINGLE SCALE'
      PRINT*,'2 = TWO PLOTS -- ONE SINGLE SCALE, ONE DOUBLE SCALE'
      PRINT*,'3 = SINGLE PLOT, DOUBLE SCALE'
 910  PRINT*,'SPECIFY TYPE OF PLOT: '
      READ*,IPSC
      IPSC=IPSC-2
      IF(IPSC.GT.1.OR.IPSC.LT.-1) GO TO 910
      CALL PLOTLP(PLTVEC,37,2,IPSC,1,0,TITLE)
C
C****************************************************************
C
C REPEAT PROGRAM UNTIL USER TERMINATION
C
C****************************************************************
C
      IF(INFLAG.NE.0) PRINT*,'POSSIBLE ERRORS...INFO FLAG = ',INFLAG
 998  GO TO 30
 999  ENDFILE(4)
      REWIND(4)
      CLOSE(4)
C
C END OF MAIN PROGRAM CGTSVD------------------------------------------
C
      END
C
C****************************************************************
C****************************************************************
C
      SUBROUTINE CREAD(A,L,M,AA)
C
C****************************************************************
C
C THIS ROUTINE WILL PROMPT THE USER TO INPUT AN L BY M REAL MATRIX
C FROM THE KEYBOARD.  THE MATRIX IS THEN CHANGED TO TYPE COMPLEX, AND
C RETURNED AS A.  AA IS A DUMMY REAL ARRAY.
C
C****************************************************************
C
      COMPLEX A(16,*)
      REAL AA(M)
      INTEGER I,J,L,M
      PRINT'(" ",I2,1X,"BY",1X,I2)',L,M
      DO 100 I=1,L
         PRINT*,'ENTER ROW',I,': '
         READ*,(AA(J),J=1,M)
            DO 100 J=1,M
            A(I,J)=CMPLX(AA(J),0.0)
 100  CONTINUE
      END
C
C END SUBROUTINE CREAD ------------------------------------------
C
C
      SUBROUTINE CMATIN(AA,B,M,A,MFLAG)
C
C****************************************************************
C
```

```
C THIS ROUTINE WILL FIND THE INVERSE OF A COMPLEX SQUARE MATRIX OF
C OF DIMENSION M.
C AA=INPUT MATRIX, RETURNED UNHARMED
C B= OUTPUT MATRIX
C A= A DUMMY ARRAY
C MFLAG=SET TO 1 IF MATRIX IS ALGORITHMICALLY SINGULAR
C
C******************************************************************
C
      COMPLEX AA(16,*),A(M,M),B(16,*),TMAT
      INTEGER I,J,K,L,M,MAXP,MFLAG
      MFLAG=0
      DO 50 I=1,M
         DO 40 J=1,M
            A(I,J)=AA(I,J)
            B(I,J)=CMPLX(0.0,0.0)
 40      CONTINUE
         B(I,I)=CMPLX(1.0,0.0)
 50   CONTINUE
      DO 600 J=1,M
         MAXP=J
         DO 200 I=J,M
            IF(CABS(A(I,J)).GT.CABS(A(MAXP,J))) THEN
               MAXP=I
            END IF
 200     CONTINUE
         IF(CABS(A(MAXP,J)).LT.1.0E-12)THEN
            PRINT*,'SINGULAR MATRIX IN CMATIN'
            MFLAG=1
            RETURN
         ELSE
            DO 300 L=1,M
               TMAT=A(J,L)
               A(J,L)=A(MAXP,L)
               A(MAXP,L)=TMAT
               TMAT=B(J,L)
               B(J,L)=B(MAXP,L)
               B(MAXP,L)=TMAT
 300        CONTINUE
            TMAT=A(J,J)
            DO 400 L=1,M
               A(J,L)=A(J,L)/TMAT
               B(J,L)=B(J,L)/TMAT
 400        CONTINUE
            DO 550 L=1,M
               IF(J-L)450,550,450
 450           TMAT=A(L,J)
               DO 500 K=1,M
                  A(L,K)=A(L,K)-A(J,K)*TMAT
                  B(L,K)=B(L,K)-B(J,K)*TMAT
 500           CONTINUE
 550        CONTINUE
         END IF
 600  CONTINUE
```

```
      END
C
C END SUBROUTINE CMATIN -------------------------------------------
C
C
      SUBROUTINE CMATML(A,B,C,L,M,N)
C
C*****************************************************************
C
C THIS ROUTINE WILL MULTIPLY TWO COMPLEX MATRICES
C A=AN L BY M COMPLEX MATRIX
C B=AN M BY N COMPLEX MATRIX
C C=THE L BY N COMPLEX PRODUCT OF A AND B
C NOTE: ACTUAL ARGUMENT C MUST DIFFER FROM A AND B
C
C*****************************************************************
C
      COMPLEX A(16,*),B(16,*),C(16,*)
      INTEGER I,J,K,L,M,N
      DO 100 I=1,L
         DO 100 J=1,N
            C(I,J)=CMPLX(0.0,0.0)
  100 CONTINUE
      DO 200 I=1,L
         DO 200 J=1,N
            DO 200 K=1,M
               C(I,J)=C(I,J)+A(I,K)*B(K,J)
  200 CONTINUE
      END
C
C END SUBROUTINE CMATML -------------------------------------------
C
C
      SUBROUTINE CMATAD(A,B,C,L,M)
C
C*****************************************************************
C
C THIS ROUTINE ADDS TWO COMPLEX MATRICES OF DIMENSION L BY M
C A AND B ARE THE INPUTS, C IS THE SUM
C
C*****************************************************************
C
      COMPLEX A(16,*),B(16,*),C(16,*)
      INTEGER I,J,L,M
      DO 100 I=1,L
         DO 100 J=1,M
            C(I,J)=A(I,J)+B(I,J)
  100 CONTINUE
      END
C
C END SUBROUTINE CMATAD -------------------------------------------
C
C
      SUBROUTINE CMATSB(A,B,C,L,M)
```

```
C
C*********************************************************************
C
C THIS ROUTINE SUBTRACTS COMPLEX MATRIX B FROM COMPLEX MATRIX A
C DIFFERENCE IS RETURNED IN COMPLEX MATRIX C.
C ALL THREE MATRICES ARE OF DIMENSION L BY M
C
C*********************************************************************
C
      COMPLEX A(16,*),B(16,*),C(16,*)
      INTEGER I,J,L,M
      DO 100 I=1,L
         DO 100 J=1,M
            C(I,J)=A(I,J)-B(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE CMATSB ----------------------------------------------
C
C
      SUBROUTINE CSMUL(A,B,C,L,M)
C
C*********************************************************************
C
C THIS ROUTINE MULTIPLIES A COMPLEX MATRIX BY A COMPLEX SCALAR
C A= THE COMPLEX SCALAR
C B= THE COMPLEX MATRIX
C C= THE COMPLEX PRODUCT
C B AND C ARE OF DIMENSION L BY M
C
C*********************************************************************
C
      COMPLEX A,B(16,*),C(16,*)
      INTEGER I,J,L,M
      DO 100 I=1,L
         DO 100 J=1,M
            C(I,J)=A*B(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE CSMUL -----------------------------------------------
C
C
      SUBROUTINE COPYMT(A,B,N,M)
C
C*********************************************************************
C
C THIS ROUTINE COPIES COMPLEX MATRIX A INTO COMPLEX MATRIX B.
C BOTH MATRICES ARE OF DIMENSION N BY M.
C
C*********************************************************************
C
      COMPLEX A(16,*),B(16,*)
      INTEGER I,J,N,M
      DO 100 I=1,N
         DO 100 J=1,M
            B(I,J)=A(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE COPYMT ----------------------------------------------
C
C
      SUBROUTINE CMTOUT(A,M,N)
C
C**********************************************************************
C
C THIS ROUTINE OUTPUTS AN M BY N COMPLEX MATRIX A
C
C**********************************************************************
C
      COMPLEX A(16,*)
      INTEGER I,J,M,N
      PRINT*
      DO 100 I=1,M
         PRINT'(" ",3(2(E9.3,2X),2X))',(A(I,J),J=1,N)
         PRINT*
 100  CONTINUE
      END
C
C END SUBROUTINE CMTOUT ----------------------------------------------
C
C
      SUBROUTINE RPOUT(A,SCRACH,M,N)
C
C**********************************************************************
C
C THIS ROUTINE PRINTS OUT THE REAL PARTS OF A COMPLEX MATRIX A
C
C*********************************************************************
C
      COMPLEX A(16,*)
      REAL SCRACH(16,*)
      INTEGER I,J,N,M
      DO 100 I=1,M
         DO 100 J=1,N
            SCRACH(I,J)=REAL(A(I,J))
 100  CONTINUE
      DO 200 I=1,M
         PRINT'(" ",5(E11.4,3X))',(SCRACH(I,J),J=1,N)
         PRINT*
 200  CONTINUE
      END
C
C END SUBROUTINE RPOUT -----------------------------------------------
C
C
      SUBROUTINE CDSCRT(A,N,TSAMP,CPHI,CPHINT,M,TP,TIDENT,CWORK)
```

```
9280=C
9290=C*****************************************************************
9300=C
9310=C THIS ROUTINE APPROXIMATES THE STATE TRANSITION MATRIX AND ITS
9320=C INTEGRAL FOR A TIME INVARIANT LINEAR SYSTEM AS A MATRIX EXPONENTIAL
9330=C OVER A SMALL SAMPLE PERIOD.  RESULTS RETURNED IN COMPLEX MATRICES.
9340=C A= SYSTEM DYNAMICS MATRIX, TYPE COMPLEX
9350=C N= STATE DIMENSION
9360=C TSAMP= SAMPLING PERIOD
9370=C CPHI= STATE TRANSITION MATRIX, TYPE COMPLEX
9380=C CPHINT= APPROXIMATE INTEGRAL OF CPHI, TYPE COMPLEX
9390=C M= NUMBER OF TERMS USED IN EXPONENTIAL EXPANSION
9400=C TP, TIDENT AND CWORK ARE DUMMY ARRAYS
9410=C
9420=C*****************************************************************
9430=C
9440=      COMPLEX A(16,*),CPHINT(16,*),CPHI(16,*),CT,TIDENT(16,*),TP(16,*)
9450=      COMPLEX CWORK(16,*),CRIJ
9460=      REAL TSAMP,RIJ
9470=      INTEGER I,J,M,N
9480=      CT=CMPLX(TSAMP,0.0)
9490=      DO 200 I=1,N
9500=         DO 100 J=1,N
9510=            TIDENT(I,J)=CMPLX(0.0,0.0)
9520= 100     CONTINUE
9530=         TIDENT(I,I)=CMPLX(1.0,0.0)
9540= 200  CONTINUE
9550=      CALL CSMUL(CT,TIDENT,CPHINT,N,N)
9560=      CALL COPYMT(CPHINT,TP,N,N)
9570=      CALL CSMUL(CT,A,CPHI,N,N)
9580=      DO 300 I=1,M
9590=         CALL CMATML(TP,CPHI,CWORK,N,N,N)
9600=        'CALL COPYMT(CWORK,TP,N,N)
9610=         RIJ=1.0/REAL(I+1)
9620=         CRIJ=CMPLX(RIJ,0.0)
9630=         CALL CSMUL(CRIJ,TP,TP,N,N)
9640=         CALL CMATAD(CPHINT,TP,CPHINT,N,N)
9650= 300  CONTINUE
9660=      CALL CMATML(A,CPHINT,TP,N,N,N)
9670=      CALL CMATAD(TIDENT,TP,CPHI,N,N)
9680=      END
9690=C
9700=C END SUBROUTINE CDSCRT ------------------------------------------
9710=C
9720=C
9730=      SUBROUTINE SETPLT(A,N,M,X)
9740=C
9750=C*****************************************************************
9760=C
9770=C THIS ROUTINE CONVERTS A REAL MATRIX OF DIMENSION N BY M INTO A
9780=C VECTOR THAT IS COMPATIBLE WITH R.M. FLOYD'S PRINTER PLOTTING
9790=C ROUTINE, PLOTLP.  THE INPUT MATRIX IS A.
9800=C N= ROW DIMENSION OF A, THE NUMBER OF POINTS TO BE PLOTTED
9810=C M= COLUMN DIMENSION OF A, THE NUMBER OF FUNCTIONS TO BE PLOTTED +1
```

```
C X= THE PLOTTING VECTOR, DIMENSION N*M
C
C*********************************************************************
C
      REAL A(N,M),X(N*M)
      INTEGER N,M,I,J
      DO 100 J=1,M
         DO 100 I=1,N
            X(I+(J-1)*N)=A(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE SETPLT ---------------------------------------------
C
C
      SUBROUTINE PLOTLP(A,N,M,IPSC,ISCL,LPTERM,TITLE)
C
C*********************************************************************
C
C THIS ROUTINE WAS ADAPTED FROM R.M. FLOYD'S THESIS TO PRODUCE
C PRINTER PLOTS OF COMPUTED RESULTS.
C A= VECTOR OF DATA, CONVERTED FROM MATRIX FORM BY SUBROUTINE SETPLT
C N= NUMBER OF POINTS (INDEPENDENT VARIABLE) TO BE PLOTTED
C M= NUMBER OF FUNCTIONS (DEPENDENT VARIABLES) TO BE PLOTTED
C IPSC = -1-->ALL VARIABLES SCALED TOGETHER (1 PLOT)
C      =  0-->SCALED TOGETHER AND SEPARATELY (2 PLOTS)
C      = +1-->SCALED SEPARATELY (1 PLOT)
C ISCL =  0-->PLOT OVER EXACT RANGE OF VARIABLE
C        +1-->PLOT WITH EVEN SCALING
C LPTERM = 0-->PLOT 50 CHARACTERS WIDE
C         +1-->PLOT 100 CHARACTERS WIDE
C TITLE = MAX OF 50 CHARACTERS, TYPE CHARACTER
C
C*********************************************************************
C
      REAL YSCAL(6),YMIN(6),YPR(11),RISPAC,RMIN,RMAX,YL,YH,XPR,A(*)
      REAL SCAL
      INTEGER IBLNK(6),IPSC,ISCL,LPTERM,IPAPER,ISPAC,IPRTI,ISC,J,IC,IX
      INTEGER IL,JP,ITEMP,M1,M2,M,N,ICO,I
      CHARACTER TITLE*50
      CHARACTER*1 BLANK,PLUS,COLON,GRID,SYMBOL(6),OUT(101)
      DATA BLANK,PLUS,COLON,SYMBOL(1),SYMBOL(2)/' ','+',':','1','2'/
      DATA SYMBOL(3),SYMBOL(4),SYMBOL(5),SYMBOL(6)/'3','4','5','6'/
      IPAPER=5*(1+LPTERM)
      ISPAC=10*IPAPER
      RISPAC=REAL(ISPAC)
      ISPAC=ISPAC+1
      IPRTI=IPAPER+1
      RMIN=A(N+1)
      RMAX=RMIN
 25   DO 41 ISC=1,M
         M1=ISC*N+1
         YL=A(M1)
         YH=YL
         M2=N*(ISC+1)
         DO 40 J=M1,M2
            IF(A(J).LT.YL)THEN
               YL=A(J)
            END IF
            IF(A(J).GT.YH)THEN
               YH=A(J)
            END IF
 40      CONTINUE
         IF(YL.LT.RMIN)THEN
            RMIN=YL
         END IF
         IF(YH.GT.RMAX)THEN
            RMAX=YH
         END IF
         IF(IPSC.GE.0)THEN
            CALL VARSCL(YL,YH,YSCAL(ISC),RISPAC,ISCL)
         END IF
         YMIN(ISC)=YL
 41   CONTINUE
      IF(IPSC.LE.0) THEN
         CALL VARSCL(RMIN,RMAX,SCAL,RISPAC,ISCL)
      END IF
      IC=2-IABS(IPSC)
      DO 42 IX=1,ISPAC
         OUT(IX)=BLANK
 42   CONTINUE
      DO 100,ICO=1,IC
         PRINT'("1",11X,A50)',TITLE
         WRITE(4,'(11X,A50)')TITLE
         WRITE(4,'(A1)')BLANK
         PRINT*
         DO 60 I=1,N
            XPR=A(I)
            IF(MOD(I,10).EQ.0)THEN
               GRID=COLON
            ELSE
               GRID=BLANK
            END IF
            DO 44 IX=2,ISPAC,2
               OUT(IX)=GRID
 44         CONTINUE
            DO 46 IX=1,ISPAC,10
               OUT(IX)=PLUS
 46         CONTINUE
            DO 55 J=1,N
               IL=I+J*N
               IF(IPSC.EQ.-1)THEN
                 JP=INT((A(IL)-RMIN)/SCAL)+1
               ELSE IF(IPSC.EQ.0)THEN
                  IPSCT=IPSC+ICO
                  IF(IPSCT.EQ.2)THEN
                     JP=INT((A(IL)-YMIN(J))/YSCAL(J))+1
                  ELSE
```

```
                     JP=INT((A(IL)-RMIN)/SCAL)+1
                  END IF
               ELSE
                 JP=INT((A(IL)-YMIN(J))/YSCAL(J))+1
               END IF
50             OUT(JP)=SYMBOL(J)
               IBLNK(J)=JP
55          CONTINUE
            PRINT'(" ",F11.4,6X,101A1)',XPR,(OUT(IX),IX=1,ISPAC)
            WRITE(4,'(F11.4,6X,101A1)')XPR,(OUT(IX),IX=1,ISPAC)
            DO 59 J=1,M
               ITEMP=IBLNK(J)
               OUT(ITEMP)=BLANK
59          CONTINUE
60       CONTINUE
         IF(IPSC.NE.1)THEN
            IF(IPSCT.NE.2)THEN
               YPR(1)=RMIN
               DO 70 I=1,IPAPER
                  YPR(I+1)=YPR(I)+10.*SCAL
70             CONTINUE
               PRINT'("0    SCALE   ",11E10.3)',(YPR(I),I=1,IPRTI)
               WRITE(4,'(A1)')BLANK
               WRITE(4,'("    SCALE   ",11E10.3)')(YPR(I),I=1,IPRTI)
               WRITE(4,'(A1)')BLANK
               WRITE(4,'(A1)')BLANK
            END IF
         END IF
         IF(IPSC.EQ.1.OR.IPSCT.EQ.2)THEN
            DO 76 ISC=1,M
               YPR(1)=YMIN(ISC)
               DO 74 I=1,IPAPER
                  YPR(I+1)=YPR(I)+10.*YSCAL(ISC)
74             CONTINUE
               PRINT'("0    SCALE ",A1,1X,11E10.3)',SYMBOL(ISC),(YPR(IX)
     1,IX=1,IPRTI)
               WRITE(4,'(A1)')BLANK
               WRITE(4,'("    SCALE ",A1,1X,11E10.3)')SYMBOL(ISC),
     1(YPR(IX),IX=1,IPRTI)
76          CONTINUE
         END IF
         DO 90 ISC=1,60-N
            WRITE(4,'(A1)')BLANK
90       CONTINUE
100   CONTINUE
      PRINT'("1")'
      END
C
C END SUBROUTINE PLOTLP -------------------------------------------
C
C
      SUBROUTINE VARSCL(XMIN,XMAX,SCALE,RSPACE,ISCL)
C
C*******************************************************************
```

```
C
C THIS IS A SCALING ROUTINE THAT SUPPORTS PLOTLP
C ADAPTED FROM R.M. FLOYD'S THESIS
C
C*****************************************************************
C
      REAL XMIN,XMAX,SCALE,RSPACE,EXP,XMINT,XMAXT
      INTEGER ISCL,ISCAL
      IF(XMAX.EQ.XMIN)THEN
         XMIN=.9*XMIN-10.
      END IF
      SCALE=XMAX-XMIN
      IF(ISCL.NE.0)THEN
         EXP=INT(100.+LOG10(SCALE))-100.
         FACTOR=10.**(1.-EXP)
         XMINT=XMIN*FACTOR
         XMAXT=XMAX*FACTOR
         IF(XMAXT.GE.0.)THEN
            XMAXT=XMAXT+.9
         END IF
         IF(XMINT.LE.0.)THEN
            XMINT=XMINT-.9
         END IF
         XMINT=AINT(XMINT)
         ISCAL=XMAXT-XMINT
         IF(MOD(ISCAL,5).NE.0)THEN
            ISCAL=ISCAL+5-MOD(ISCAL,5)
         END IF
         FACTOR=10.**(EXP-1.)
         XMIN=XMINT*FACTOR
         SCALE=FACTOR*REAL(ISCAL)
      END IF
      SCALE=SCALE/RSPACE
      END
C
C END SUBROUTINE VARSCL --------------------------------------------
C
C
      SUBROUTINE EDIT(EDMAT,M,N,SCRACH)
C
C*****************************************************************
C
C THIS ROUTINE ALLOWS THE USER TO EDIT AN M BY N MATRIX EDMAT
C
C*****************************************************************
C
      COMPLEX EDMAT(16,*)
      REAL EL,SCRACH(16,*)
      INTEGER M,N,I,J
      CHARACTER ANSW*1
 10   PRINT*,'LIST CURRENT VALUES? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 10
      IF(ANSW.EQ.'Y') CALL RPOUT(EDMAT,SCRACH,M,N)
      PRINT*,'ENTER 0,0,0 <CR> WHEN ALL CHANGES HAVE BEEN MADE'
 100  PRINT'(" ENTER ROW #, COLUMN #, AND MATRIX ELEMENT- ",
     1 I2,1X,"BY",I2,": " )',M,N
 110  READ*,I,J,EL
      IF(I.GT.0.AND.I.LE.M.AND.J.GT.0.AND.J.LE.N)THEN
         EDMAT(I,J)=CMPLX(EL,0.0)
         GO TO 110
      ELSE IF(I.EQ.0.AND.J.EQ.0)THEN
 150  PRINT*,'LIST MODIFIED MATRIX? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 150
      IF(ANSW.EQ.'Y') CALL RPOUT(EDMAT,SCRACH,M,N)
 200     PRINT*,'ANY MORE CHANGES TO THIS MATRIX? Y/N: '
         READ(*,'(A)')ANSW
         IF(ANSW.NE.'Y'.AND.ANSW.NE.'N')GO TO 200
         IF(ANSW.EQ.'Y')GO TO 100
         IF(ANSW.EQ.'N')RETURN
      ELSE
         PRINT*,'SUBSCRIPT OUT OF RANGE'
         GO TO 100
      END IF
      END
C
C END SUBROUTINE EDIT---------------------------------------------
C
C
      SUBROUTINE CXPOSE(A,B,M,N)
C
C*****************************************************************
C
C THIS ROUTINE WILL TRANSPOSE AN M BY N COMPLEX MATRIX A
C THE RESULT IS AN N BY M COMPLEX MATRIX B
C
C*****************************************************************
C
      COMPLEX A(16,*),B(16,*)
      INTEGER I,J,M,N
      DO 100 I=1,N
         DO 100 J=1,M
            B(I,J)=A(J,I)
 100  CONTINUE
C
C END SUBROUTINE CXPOSE ------------------------------------------
      END
```

C. ODEACT Software Description and Instructions

C.1 Introduction

ODEACT is an interactive computer program that was developed to provide the capability to evaluate the controller designs of this study with respect to actuator nonlinearities (specifically saturation, for this research). The purpose of this appendix is to outline the program structure and explain its use. A sample execution and the complete source code listing are included.

ODEACT employs an integration package called "ODE" [42] to simulate the controller time response, with various system truth models, to non-zero initial conditions and command inputs. User options include the choice of one-, three-, or four-state actuator models as defined in Section 4.3; the selection of linear or rate- and position-limited nonlinear actuators; the addition of anti-windup compensation to the control law, as developed in Section 5.7; and variation of plant truth model parameters for the pitch, angle of attack and pitch rate states through redefinition of the appropriate elements of the truth model dynamics matrix.

Unlike CGTSVD, ODEACT is a special purpose program and, in its present form, is only useful for analysis of deterministic CGT/PI controller designs based on the five-state design model for the AFTI F-16 pitch-pointing system, as defined in Section 4.2. While not directly usable for analysis of other designs, ODEACT can serve as a model analysis tool which could easily be adapted to other problems. Because its purpose is so specialized, the structure of ODEACT is extremely simple. As with CGTSVD, sophisticated progamming practices were clearly unnecessary; the load size for ODEACT is only about 42,000 octal words of memory. The program is written in ANSI standard FORTRAN 77. The source code is explained by numerous comments, and the prompts during execution are sufficient in number and detail to assist the novice user; the additional comments and clarification provided in this appendix will therefore be brief.

C.2 ODEACT Program Structure

ODEACT is entirely self contained except for the use of the subroutine "ODE" [42] which is maintained in the CC6600 library on the CYBER computer. Many of the utility subroutines are simply adaptations of those used in CGTSVD, but using real variables and arithmetic, rather than complex. Also, matrix partitions of larger arrays are not used in the calculations, so row dimensions are passed as arguments to the matrix manipulation routines, eliminating the need to employ the inefficient practice of blocking all matrices with a common row dimension.

The program utilizes the basic design model dynamics description of the AFTI F-16 found in Section 4.2, except that the state dimension is expanded from 5 to 11 to allow for up to 4 states for each actuator. This means that the trailing edge flap state, which was originally state 5, becomes state 8. Since the actuator models and the control input matrix are "hardwired" into the program, this alteration is transparent to the user except when inputting the dynamics matrix ($\underline{A}$), the output matrix ($\underline{C}$) and the $\underline{K}_x$ matrix, as discussed in Section C.3.

ODEACT assumes that design parameters will be read in from a previously developed data file at line 670; if this is not the case, keyboard inputs are made at line 920. In either case, the user is permitted to change any parameters before each simulation run via the "change menu" at line 1440, and to save each set of parameters to a data file. Once the parameters have been established, the CGT command model is discretized (line 1880), the actuator model and control law options selected (line 2000), and initial conditions and input command are specified (line 2310). In ODEACT's current form, initial conditions may be applied to any state, but command inputs are limited to step inputs to the pitch angle. This can be changed by modifying the code beginning at line 2430. The plotted outputs are the pitch angle (represented by the symbol "1"), flight path angle (symbol "2"), and actuator states (symbol "3" for the horizontal tail, symbol "4" for the trailing edge flap); this can be changed by modifying the code beginning at line 2510.

The control input for each sample period is calculated by a call to subroutine "GCSTAR" at line 2670; "GCSTAR" propagates the command model to define the new command model states, and then calculates the controls as defined in (II-25). The control inputs are modified for anti-windup compensation, as defined in Section 5.7, if that option is selected. "ODE" is then called to propagate the system states over the ensuing sample period. There are three alternative calling statements for "ODE," differing only in the first argument, which is the argument that specifies the subroutine containing the selected actuator model. "F1" contains the single-state model, "F3" the three-state, and "F4" the four-state model. Each of these subroutines contains the set of ordinary differential equations that make up the appropriate dynamics description for overall controlled system which "ODE" calls repeatedly during the conduct of the integration. If nonlinear actuators are specified by the user, then rate and position limits are applied to the actuators in these subroutines. For example, if the actuator position state reaches a positive position limit, then its derivative is constrained to be non-positive. If the state derivative reaches a positive rate limit, then the next derivative is constrained to be non-positive.

The plotted output for each simulation includes 51 points, regardless of the duration of the response. The time interval between the points is an integer multiple of the sample period chosen to ensure that the specified response duration is covered by the plot. When "ODE" completes the integration over a sample period, the states are stored in the "OUT" array if the current time is a plot interval, and then the next sample period is begun with a call to "GCSTAR".

When the simulation is completed, the user is given the option to change any design parameters or exit the program through the code at line 2970. When the single-state actuator model is used, 1 second of simulation uses about 0.5 seconds of processor time on the CYBER computer. When the four-state actuator model is used, 2 to 3 seconds of processor time is required for a 1 second simulation, the longer time being applicable when actuator nonlinearities are simulated. These times apply to the program as listed in Section C.5; the amount of processor time used can be reduced by increasing the values of the error tolerances specified in lines 2480 and 2490. In fact, the tolerances listed, which were the ones used for this study, were without doubt much smaller than necessary to achieve good accuracy in the simulation. Tests with both error tolerances set at 0.001 did not visibly affect the plotted results, and provided a 50% savings in processor time with the four-state actuator model. Applicable error tolerance selection for "ODE" is a function of the problem being run, and is discussed at length in [42].

C.3 Instructions for Using ODEACT

Prior to executing ODEACT, any previously developed data files that are to be used must be attached. File usage by ODEACT is the same as that outlined in Appendix B for CGTSVD, with the exception that ODEACT will overwrite a previously established data file. Data files for ODEACT are not compatible with those for CGTSVD or CGTPIV. The library containing "ODE" (CC6600 for the Aeronautical Systems Division CYBER computer) must also be attached and declared as a library.

Once the program execution begins, the user need only respond to the program prompts. Only the non-zero elements of any matrix need to be entered. The only "trick" to using the program is to remember, when inputting the $\underline{A}$, $\underline{C}$ or $\underline{K}_x$ matrices (which go by the same names in the program output), that what would normally be the elements of the fifth column (elements tied to the trailing edge flap state) must be entered as the eighth column, regardless of which actuator model is selected. Only the first 3 rows of the $\underline{A}$ matrix are entered, since the rest represent actuator states already represented in program subroutines. The CGT command model is represented in the "AM", "BM" and "CM" matrices. The command model used must be of dimension 4 or less; internally, the program uses a four-state representation in any case, and unused elements of these arrays are simply set to zero and do not affect the results.

All of the time-response plots displayed at the terminal are also written to a plot file named by the user. Upon exit from the program, the plot file can be saved or routed to a printer to obtain a "hard" copy of the results.

C.4 Sample Program Execution

The pages that follow contain a short sample run of ODEACT on the CYBER computer. The object code is the file named ODEACT and the data file ODESRB is the data for the SR-B CGT/PI design described in Section 5.2. The computer output is displayed in upper case and the non-numeric user responses in lower case text. The message concerning a "NON-FATAL LOADER ERRORS" is not significant to ODEACT, but pertains to an unused portion of the CC6600 library.

```
COMMAND- attach,odesrb

 PFN IS
 ODESRB
 AT CY= 001 SN=AFIT

COMMAND- attach,odeact

 PFN IS
 ODEACT
 AT CY= 001 SN=AFIT

COMMAND- attach,cc6600,id=library,sn=asd

 PFN IS
 CC6600
 AT CY= 999 SN=ASD

COMMAND- library,cc6600

COMMAND- odeact

ENTER NAME FOR PLOT OUTPUT FILE: plot2
      NON-FATAL LOADER ERRORS -
 NON-EXISTENT LIBRARY GIVEN -  SYSIO

DATA TO BE READ FROM FILE? Y/N: y

ENTER NAME OF DATA FILE: odesrb

ANY CHANGES TO MATRICES? Y/N: n

WRITE DATA TO OUTPUT FILE? Y/N: n

ENTER SAMPLING TIME: .02

PHI MATRIX FOR COMMAND MODEL:
  .9048E+00    0.            0.            0.

 0.            .9048E+00    0.            0.

 0.            0.            .1000E+01    0.

 0.            0.            0.            .1000E+01
```

```
BD MATRIX FOR COMMAND MODEL:
  .9516E-01    0.            0.            0.

 0.             .9516E-01    0.            0.

 0.            0.            0.            0.

 0.            0.            0.            0.

1= FOUR STATE    2= THREE STATE   3= SINGLE STATE
SELECT ACTUATOR MODEL: 3

APPLY ACTUATOR RATE/POSITION LIMITS? Y/N: y

ENPLOY ANTI-WINDUP COMPENSATION? Y/N: y

ENTER DESIRED RESPONSE DURATION: 2

ENTER INITIAL CONDITIONS FOR STATES, IF NON-ZERO:
ENTER I AND X(I); 0,0 TO TERMINATE: 1,1
0,0

ENTER STEP INPUT MAGNITUDE FOR PITCH ANGLE: 0

+---------------ENTER TITLE FOR PLOT-------------+
sample plot for sr-b design
```

```
        SAMPLE PLOT FOR SR-0 DESIGN

 0.0000      +       3 +         +         +     1   +     4 2 +
  .0400      +4        +         +         +     1   3       2 +
  .0800      +4        +         +         +   1 3   +      2  +
  .1200      +      4  +         +3        +1        +      2  +
  .1600      +       3 +4        +      1  +         +     2    +
  .2000      +3        +   4     +  1      +         +   2     +
  .2400      3         +       4 +1        +         + 2       +
  .2800      + 3       +       1 +4        +         2         +
  .3200      +    3    +      1  +   4     +       2 +         +
  .3600      +: : : 3 :+: : 1 : :+: : :4: :+: : : 2 :+: : : : :+
  .4000      +       3 +   1     +       4 +    2    +         +
  .4400      +       3 +  1      +        4+  2      +         +
  .4800      +       3 + 1       +        4+ 2       +         +
  .5200      +      3  +1        +         4         +         +
  .5600      +     3   1         +       2 +4        +         +
  .6000      +     3  1+         +      2  + 4       +         +
  .6400      +     3 1 +         +    2    +  4      +         +
  .6800      +     31  +         +   2     +  4      +         +
  .7200      +      3  +         + 2       +   4     +         +
  .7600      +: : :13 :+: : : : :+2 : : : :+: : 4 : :+: : : : :+
  .8000      +     13  +        2+         +     4   +         +
  .8400      +    1 3  +       2 +         +      4  +         +
  .8800      +    1 3  +      2  +         +      4  +         +
  .9200      +   1  3  +    2    +         +       4 +         +
  .9600      +   1  3  +   2     +         +       4 +         +
 1.0000      +   1  3  +  2      +         +        4+         +
 1.0400      +  1   3  + 2       +         +        4+         +
 1.0800      +  1   3  +2        +         +         4         +
 1.1200      +  1   3  2         +         +         4         +
 1.1600      +: 1 : 3 2+: : : : :+: : : : :+: : : : :+4 : : : :+
 1.2000      +  1   32 +         +         +         +4        +
 1.2400      +  1   32 +         +         +         + 4       +
 1.2800      +  1   3  +         +         +         + 4       +
 1.3200      + 1   23  +         +         +         + 4       +
 1.3600      + 1  2 3  +         +         +         +  4      +
 1.4000      + 1  2 3  +         +         +         +  4      +
 1.4400      + 1 2  3  +         +         +         +  4      +
 1.4800      + 1 2  3  +         +         +         +  4      +
 1.5200      + 12   3  +         +         +         +   4     +
 1.5600      +:12 : 3 :+: : : : :+: : : : :+: : : : :+: :4: : :+
 1.6000      + 2    3  +         +         +         +   4     +
 1.6400      + 2    3  +         +         +         +   4     +
 1.6800      + 2    3  +         +         +         +   4     +
 1.7200      +21    3  +         +         +         +    4    +
 1.7600      +21    3  +         +         +         +    4    +
 1.8000      +21    3  +         +         +         +    4    +
 1.8400      +21    3  +         +         +         +    4    +
 1.8800      2 1    3  +         +         +         +    4    +
 1.9200      2 1    3  +         +         +         +    4    +
 1.9600      2:1: : 3 :+: : : : :+: : : : :+: : : : :+: : 4 : :+
 2.0000      2 1    3  +         +         +         +    4    +

SCALE 1  -.100E+00  .200E+00  .500E+00  .800E+00  .110E+01  .140E+01

SCALE 2   .300E-01  .230E+00  .430E+00  .630E+00  .830E+00  .103E+01

SCALE 3  -.400E+00  .100E+00  .600E+00  .110E+01  .160E+01  .210E+01

SCALE 4  -.140E+01 -.110E+01 -.800E+00 -.500E+00 -.200E+00  .100E+00
```

```
MORE TIME RESPONSE RUNS WITH THIS MODEL? Y/N: n

CHANGE MATRICES? Y/N: n

MORE RUNS WITH NEW MODEL? Y/N: n

     END  ODEACT
     42200 MAXIMUM EXECUTION FL.
     1.198 CP SECONDS EXECUTION TIME.

COMMAND- route,plot2,dc=pr,st=csa,tid=af,fid=wge
```

C.5 ODEACT Source Listing

```
      PROGRAM ODEACT
C
C******************************************************************
C
C SIMULATION PROGRAM TO TEST A PI REGULATOR OR CGT/PI PITCH-POINTING
C CONTROLLER BASED ON A 5-STATE MODEL OF THE AFTI F-16.  OPTIONS
C INCLUDE MODIFICATION OF DYNAMICS MATRIX, USE OF 1-, 3- OR 4-STATE
C ACTUATOR MODELS, APPLICATION OF RATE/POSITION LIMITS ON ACTUATORS,
C AND EMPLOYMENT OF ANTI-WINDUP COMPENSATION.  USER SUPPLIES DYNAMICS
C MATRIX, OUTPUT MATRIX, CGT COMMAND MODEL AND CONTROLLER GAINS WITH
C THE FOLLOWING CHANGES:
C    1.  ONLY THE FIRST THREE ROWS OF THE DYNAMICS MATRIX ARE ENTERED
C    2.  COLUMN 5 OF THE NORMAL DYNAMICS MATRIX IS ENTERED AS COLUMN 8
C    3.  COLUMN 5 OF THE NORMAL KX MATRIX IS ENTERED AS COLUMN 8
C THE COMMAND MODEL MUST BE OF DIMENSION 4 OR LESS
C
C DATE OF LAST REVISION:  6 SEP 83
C LIBRARIES USED:  CC6600,ID=LIBRARY,SN=ASD
C
C******************************************************************
C
      REAL WORK(350),X(11),DX(11),OUT(51,5),T,TOUT,TSAMP,PLTVEC(260)
      REAL AWORK(4,4),BWORK(4,4),CWORK(4,4),AM(4,4),BM(4,4)
      REAL RELERR,ABSERR,DSIM
      INTEGER I,J,K,IFLAG,JFLAG,JCFLAG,KFLAG,NFLAG,IWORK(5),IDSIM
      COMMON/MATRIX/A(3,11),C(4,11),KX(2,11),KZ(2,4),KXM(2,4),
     1 KXU(2,4),PHI(4,4),PHINT(4,4),CM(4,4)
      COMMON/CONTRL/UNEW(2),UOLD(2),UCMD(4),UCOLD(4),XOLD(11),
     1 XMOLD(4),XM(4),MFLAG
      EXTERNAL F1,F3,F4
      CHARACTER ANSW*1,TITLE*50,DATA*6,SAVE*6,PLOT*6
C
C******************************************************************
C
C FLAG DEFINITIONS:
C      IFLAG = INSTRUCTIONS FOR ODE INTEGRATION SUBROUTINE; INPUT MUST
C              BE NEGATIVE, OUTPUT OTHER THAN 2 SIGNALS INTEGRATION
C              PROBLEMS.
C      JFLAG = MATRIX MODIFICATION IN PROGRESS IF NON-ZERO
C      JCFLAG = CURRENT COMMAND MODEL HAS BEEN DISCRETIZED IF JCFLAG=1
C      KFLAG = CHOICE OF ACTUATOR MODEL TO BE USED
C      MFLAG = RATE/POSITION LIMITS APPLIED IF MFLAG=1
C      NFLAG = ANTI-WINDUP COMPENSATION EMPLOYED IF NFLAG=1
C
C******************************************************************
C
C******************************************************************
C
C INPUT SECTION.  DATA MAY BE READ IN FROM AN 'OLD' FILE, AND SAVED
C TO ANY OTHER FILE.  ONLY ONE SET OF DATA PER FILE NAME.  PLOTS ARE
C AUTOMATICALLY SAVED IN A 'PLOT FILE'.
C
C******************************************************************
C
```

```
 10   PRINT*,'ENTER NAME FOR PLOT OUTPUT FILE: '
      READ(*,'(A)')PLOT
      OPEN(4,FILE=PLOT,STATUS='NEW',FORM='FORMATTED',ERR=10)
 20   PRINT*,'DATA TO BE READ FROM FILE? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 20
      IF(ANSW.EQ.'N') GO TO 30
         JCFLAG=0
         PRINT*,'ENTER NAME OF DATA FILE: '
         READ(*,'(A)')DATA
         OPEN(2,FILE=DATA,STATUS='OLD',FORM='UNFORMATTED',ERR=20)
         READ(2)((A(I,J),I=1,3),J=1,11)
         READ(2)((C(I,J),I=1,4),J=1,11)
         READ(2)((KX(I,J),I=1,2),J=1,11)
         READ(2)((KZ(I,J),I=1,2),J=1,4)
         READ(2)((KXU(I,J),I=1,2),J=1,4)
         READ(2)((KXM(I,J),I=1,2),J=1,4)
         READ(2)((AM(I,J),I=1,4),J=1,4)
         READ(2)((BM(I,J),I=1,4),J=1,4)
         READ(2)((CM(I,J),I=1,4),J=1,4)
         REWIND(2)
         CLOSE(2)
      GO TO 140
C
C FOR KEYBOARD INPUT, ONLY NON-ZERO MATRIX ELEMENTS ARE REQUIRED.
C NO NON-ZERO ENTRIES SHOULD BE MADE FOR COLUMNS 5,6,7,9,10 OR 11
C OF A OR KX, BUT NO PROTECTION PROVIDED AGAINST DOING SO.
C
 30   DO 50 I=1,3
         DO 50 J=1,11
            A(I,J)=0.0
 50   CONTINUE
      DO 64 I=1,11
         DO 60 J=1,2
            KX(J,I)=0.0
 60      CONTINUE
         DO 64 K=1,4
            C(K,I)=0.0
 64   CONTINUE
      DO 66 I=1,2
         DO 66 J=1,4
            KZ(I,J)=0.0
            KXU(I,J)=0.0
            KXM(I,J)=0.0
 66   CONTINUE
      DO 70 I=1,4
         DO 70 J=1,4
            AM(I,J)=0
            BM(I,J)=0
            CM(I,J)=0
 70   CONTINUE
      JFLAG=0
 72   PRINT*,'ENTER DYNAMICS MATRIX: '
      CALL EDIT(A,3,11)
      IF(JFLAG.NE.0) GO TO 140
   76 PRINT*,'ENTER OUTPUT MATRIX: '
      CALL EDIT(C,4,11)
      IF(JFLAG.NE.0) GO TO 140
   78 PRINT*,'ENTER KX MATRIX: '
      CALL EDIT(KX,2,11)
      IF(JFLAG.NE.0) GO TO 140
   80 PRINT*,'ENTER KZ MATRIX: '
      CALL EDIT(KZ,2,4)
      IF(JFLAG.NE.0) GO TO 140
   82 PRINT*,'ENTER KXM MATRIX: '
      CALL EDIT(KXM,2,4)
      IF(JFLAG.NE.0) GO TO 140
   84 PRINT*,'ENTER KXU MATRIX: '
      CALL EDIT(KXU,2,4)
      IF(JFLAG.NE.0) GO TO 140
   86 PRINT*,'ENTER MODEL DYNAMICS MATRIX: '
      JCFLAG=0
      CALL EDIT(AM,4,4)
      IF(JFLAG.NE.0) GO TO 140
   88 PRINT*,'ENTER MODEL CONTROL MATRIX: '
      JCFLAG=0
      CALL EDIT(BM,4,4)
      IF(JFLAG.NE.0) GO TO 140
   90 PRINT*,'ENTER MODEL OUTPUT MATRIX: '
      CALL EDIT(CM,4,4)
  140 PRINT*,'ANY CHANGES TO MATRICES? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 140
  142 IF(ANSW.EQ.'Y') THEN
         PRINT*,'1=A     2=C    3=KX    4=KZ    5=KXM     6=KXU'
         PRINT*,'7=AM    8=BM    9=CM    ENTER CHOICE: '
         READ*,JFLAG
         GO TO (72,76,78,80,82,84,86,88,90) JFLAG
      ELSE
         JFLAG=0
      END IF
  150 PRINT*,'WRITE DATA TO OUTPUT FILE? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 150
      IF(ANSW.EQ.'Y') THEN
         PRINT*,'ENTER NAME OF OUTPUT FILE: '
         READ(*,'(A)')SAVE
         OPEN(3,FILE=SAVE,FORM='UNFORMATTED',ERR=80)
         WRITE(3)((A(I,J),I=1,3),J=1,11)
         WRITE(3)((C(I,J),I=1,4),J=1,11)
         WRITE(3)((KX(I,J),I=1,2),J=1,11)
         WRITE(3)((KZ(I,J),I=1,2),J=1,4)
         WRITE(3)((KXU(I,J),I=1,2),J=1,4)
         WRITE(3)((KXM(I,J),I=1,2),J=1,4)
         WRITE(3)((AM(I,J),I=1,4),J=1,4)
         WRITE(3)((BM(I,J),I=1,4),J=1,4)
         WRITE(3)((CM(I,J),I=1,4),J=1,4)
         ENDFILE(3)
```

```
            REWIND(3)
            CLOSE(3)
         END IF
C
C*************************************************************
C
C NOW SET UP CONDITIONS FOR CALLING ODE.  ALL INITIAL CONDITIONS
C ARE ZERO UNLESS CHANGED BY USER INPUT.  ASSUMED CGT INPUT
C IS STEP COMMAND ON PITCH STATE.  PLOTTED OUTPUTS ARE:
C     1 = PITCH ANGLE
C     2 = FLIGHT PATH ANGLE
C     3 = HORIZONTAL TAIL POSITION
C     4 = TRAILING EDGE FLAP POSITION
C
C*************************************************************
C
      IF(JCFLAG.EQ.0) THEN
         PRINT*,'ENTER SAMPLING TIME: '
         READ*,TSAMP
         CALL DSCRT(AM,4,TSAMP,PHI,PHINT,30,AWORK,BWORK,CWORK)
         CALL MATML(PHINT,BM,AWORK,4,4,4)
         CALL COPYMT(AWORK,PHINT,4,4)
         PRINT*,'PHI MATRIX FOR COMMAND MODEL: '
         CALL RPOUT(PHI,4,4)
         PRINT*,'BD MATRIX FOR COMMAND MODEL: '
         CALL RPOUT(PHINT,4,4)
         JCFLAG=1
      END IF
 152  PRINT*,'1= FOUR STATE    2= THREE STATE   3= SINGLE STATE'
      PRINT*,'SELECT ACTUATOR MODEL: '
      READ*,KFLAG
      IF(KFLAG.GT.3.OR.KFLAG.LT.1) GO TO 152
 154  PRINT*,'APPLY ACTUATOR RATE/POSITION LIMITS? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 154
      IF(ANSW.EQ.'Y') MFLAG=1
      IF(ANSW.EQ.'N') MFLAG=0
 156  PRINT*,'EMPLOY ANTI-WINDUP COMPENSATION? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 156
      IF(ANSW.EQ.'Y') NFLAG=1
      IF(ANSW.EQ.'N') NFLAG=0
 158  PRINT*,'ENTER DESIRED RESPONSE DURATION: '
      READ*,DSIM
      IF(DSIM.LT.0.1) GO TO 520
      IDSIM=INT(DSIM/(50.0*TSAMP)+.99)
 160  DO 170 I=1,11
         X(I)=0.0
         XOLD(I)=0.0
 170  CONTINUE
      DO 172 I=1,2
         UOLD(I)=0.0
 172  CONTINUE
      DO 175 I=1,4
```

```
         UCMD(I)=0.0
         UCOLD(I)=0.0
         XM(I)=0.0
         XMOLD(I)=0.0
175   CONTINUE
      PRINT*,'ENTER INITIAL CONDITIONS FOR STATES, IF NON-ZERO: '
180   PRINT*,'ENTER I AND X(I); 0,0 TO TERMINATE: '
190   READ*,I,EL
      IF(I.LE.11.AND.I.GE.1) THEN
         X(I)=EL
         GO TO 190
      ELSE IF(I.EQ.0) THEN
         GO TO 200
      ELSE
         PRINT*,'SUBSCRIPT OUT OF RANGE'
         GO TO 180
      END IF
200   PRINT*,'ENTER STEP INPUT MAGNITUDE FOR PITCH ANGLE: '
      READ*,UCMD(1)
      T=0.0
      TOUT=0.0
      IFLAG=-1
      RELERR=1.E-8
      ABSERR=1.E-6
      DO 300 I=IDSIM+1,51*IDSIM+1
         IF(IDSIM.NE.1.AND.MOD(I,IDSIM).EQ.1) THEN
            J=INT(I/IDSIM)
            OUT(J,1)=TOUT
            OUT(J,2)=X(1)
            OUT(J,3)=X(1)-X(2)
            OUT(J,4)=X(4)
            OUT(J,5)=X(8)
         ELSE IF(IDSIM.EQ.1) THEN
            J=I-IDSIM
            OUT(J,1)=TOUT
            OUT(J,2)=X(1)
            OUT(J,3)=X(1)-X(2)
            OUT(J,4)=X(4)
            OUT(J,5)=X(8)
         END IF
         TOUT=TOUT+TSAMP
         CALL GCSTAR(X,NFLAG)
         DO 250 J=1,11
            XOLD(J)=X(J)
250      CONTINUE
         DO 260 J=1,2
            UOLD(J)=UNEW(J)
260      CONTINUE
         DO 262 J=1,4
            UCOLD(J)=UCMD(J)
            XMOLD(J)=XM(J)
262      CONTINUE
         IF(KFLAG.EQ.1) THEN
            CALL ODE(F4,11,X,T,TOUT,RELERR,ABSERR,IFLAG,WORK,IWORK)
         ELSE IF(KFLAG.EQ.2) THEN
            CALL ODE(F3,11,X,T,TOUT,RELERR,ABSERR,IFLAG,WORK,IWORK)
         ELSE
            CALL ODE(F1,11,X,T,TOUT,RELERR,ABSERR,IFLAG,WORK,IWORK)
         END IF
         T=TOUT
         IF(IFLAG.NE.2) THEN
            PRINT'(" IFLAG = ",I2)',IFLAG
         ELSE
            IFLAG=-2
         END IF
 300  CONTINUE
      CALL SETPLT(OUT,51,5,PLTVEC)
      PRINT*,'+--------------ENTER TITLE FOR PLOT------------+'
      PRINT*
      READ(*,'(A)')TITLE
      CALL PLOTLP(PLTVEC,51,4,1,1,0,TITLE)
 520  PRINT*,'MORE TIME RESPONSE RUNS WITH THIS MODEL? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 520
      IF(ANSW.EQ.'Y') GO TO 152
 525  PRINT*,'CHANGE MATRICES? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 525
      IF(ANSW.EQ.'Y') GO TO 142
 530  PRINT*,'MORE RUNS WITH NEW MODEL? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 530
      IF(ANSW.EQ.'Y') GO TO 20
      ENDFILE(4)
      REWIND(4)
      CLOSE(4)
      END
C
C END PROGRAM ODEACT -----------------------------------------------
C
C
      SUBROUTINE F4(T,X,DX)
C
C*******************************************************************
C
C THIS IS THE SET OF FIRST ORDER ODE THAT DEFINE THE FOUR-STATE
C ACTUATOR MODEL.  RATE AND POSITION LIMITS INCLUDED IF MFLAG=1.
C
C*******************************************************************
C
      REAL T,X(11),DX(11)
      COMMON/MATRIX/A(3,11),C(4,11),KX(2,11),KZ(2,4),KXM(2,4),
     1 KXU(2,4),PHI(4,4),PHINT(4,4),CM(4,4)
      COMMON/CONTRL/UNEW(2),UOLD(2),UCMD(4),UCOLD(4),XOLD(11),
     1 XMOLD(4),XM(4),MFLAG
      DX(1)=A(1,3)*X(3)
      DX(2)=A(2,1)*X(1)+A(2,2)*X(2)+A(2,3)*X(3)+A(2,4)*X(4)+A(2,8)*X(8)
      DX(3)=A(3,2)*X(2)+A(3,3)*X(3)+A(3,4)*X(4)+A(3,8)*X(8)
      DX(4)=X(5)
      DX(5)=X(6)
      DX(6)=X(7)
      DX(7)=-14911329.1*X(4)-1148576.7*X(5)-25364.42*X(6)-270.1*X(7)
     1 +14911329.1*UNEW(1)
      DX(8)=X(9)
      DX(9)=X(10)
      DX(10)=X(11)
      DX(11)=-14911329.1*X(8)-1148576.7*X(9)-25364.42*X(10)
     1 -270.1*X(11)+14911329.1*UNEW(2)
      IF(MFLAG.EQ.1) THEN
         IF(X(4).GE.25.0.AND.DX(4).GT.0.0) DX(4)=0.0
         IF(X(4).LE.-25.0.AND.DX(4).LT.0.0) DX(4)=0.0
         IF(X(5).GE.60.0.AND.DX(5).GT.0.0) DX(5)=0.0
         IF(X(5).LE.-60.0.AND.DX(5).LT.0.0) DX(5)=0.0
         IF(X(8).GE.20.0.AND.DX(8).GT.0.0) DX(8)=0.0
         IF(X(8).LE.-23.0.AND.DX(8).LT.0.0) DX(8)=0.0
         IF(X(9).GE.52.0.AND.DX(9).GT.0.0) DX(9)=0.0
         IF(X(9).LE.-52.0.AND.DX(9).LT.0.0) DX(9)=0.0
      END IF
      RETURN
      END
C
C END SUBROUTINE F4 ------------------------------------------------
C
C
      SUBROUTINE F3(T,X,DX)
C
C**********************************************************************
C
C THIS IS THE SET OF FIRST ORDER ODE THAT DEFINES THE THREE-STATE
C ACTUATOR MODEL.  RATE AND POSITION LIMITS ARE INCLUDED IF MFLAG=1.
C
C**********************************************************************
C
      REAL T,X(11),DX(11)
      COMMON/MATRIX/A(3,11),C(4,11),KX(2,11),KZ(2,4),KXM(2,4),
     1 KXU(2,4),PHI(4,4),PHINT(4,4),CM(4,4)
      COMMON/CONTRL/UNEW(2),UOLD(2),UCMD(4),UCOLD(4),XOLD(11),
     1 XMOLD(4),XM(4),MFLAG
      DX(1)=A(1,3)*X(3)
      DX(2)=A(2,1)*X(1)+A(2,2)*X(2)+A(2,3)*X(3)+A(2,4)*X(4)+A(2,8)*X(8)
      DX(3)=A(3,2)*X(2)+A(3,3)*X(3)+A(3,4)*X(4)+A(3,8)*X(8)
      DX(4)=X(5)
      DX(5)=X(6)
      DX(6)=-102978.792*X(4)-7220.98*X(5)-125.3*X(6)
     1 +102978.792*UNEW(1)
      DX(7)=0.0
      DX(8)=X(9)
      DX(9)=X(10)
      DX(10)=-102978.792*X(8)-7220.98*X(9)-125.3*X(10)
     1 +102978.792*UNEW(2)
      DX(11)=0.0
      IF(MFLAG.EQ.1) THEN
```

```
         IF(X(4).GE.25.0.AND.DX(4).GT.0.0) DX(4)=0.0
         IF(X(4).LE.-25.0.AND.DX(4).LT.0.0) DX(4)=0.0
         IF(X(5).GE.60.0.AND.DX(5).GT.0.0) DX(5)=0.0
         IF(X(5).LE.-60.0.AND.DX(5).LT.0.0) DX(5)=0.0
         IF(X(8).GE.20.0.AND.DX(8).GT.0.0) DX(8)=0.0
         IF(X(8).LE.-23.0.AND.DX(8).LT.0.0) DX(8)=0.0
         IF(X(9).GE.52.0.AND.DX(9).GT.0.0) DX(9)=0.0
         IF(X(9).LE.-52.0.AND.DX(9).LT.0.0) DX(9)=0.0
      END IF
      RETURN
      END
C
C END SUBROUTINE F3 -----------------------------------------------
C
C
      SUBROUTINE F1(T,X,DX)
C
C*****************************************************************
C
C THIS IS THE SET OF FIRST ORDER ODE THAT DEFINES THE SINGLE-STATE
C ACTUATOR MODEL.  POSITION AND RATE LIMITS ARE INCLUDED IF MFLAG=1.
C
C*****************************************************************
C
      REAL T,X(11),DX(11)
      COMMON/MATRIX/A(3,11),C(4,11),KX(2,11),KZ(2,4),KXM(2,4),
     1 KXU(2,4),PHI(4,4),PHINT(4,4),CM(4,4)
      COMMON/CONTRL/UNEW(2),UOLD(2),UCMD(4),UCOLD(4),XOLD(11),
     1 XMOLD(4),XM(4),MFLAG
      DX(1)=A(1,3)*X(3)
      DX(2)=A(2,1)*X(1)+A(2,2)*X(2)+A(2,3)*X(3)+A(2,4)*X(4)+A(2,8)*X(8)
      DX(3)=A(3,2)*X(2)+A(3,3)*X(3)+A(3,4)*X(4)+A(3,8)*X(8)
      DX(4)=-20.*X(4)+20.0*UNEW(1)
      DX(5)=0.0
      DX(6)=0.0
      DX(7)=0.0
      DX(8)=-20.*X(8)+20.0*UNEW(2)
      DX(9)=0.0
      DX(10)=0.0
      DX(11)=0.0
      IF(MFLAG.EQ.1) THEN
         IF(X(4).GE.25.0.AND.DX(4).GT.0.0) DX(4)=0.0
         IF(X(4).LE.-25.0.AND.DX(4).LT.0.0) DX(4)=0.0
         IF(DX(4).GT.60.0) DX(4)=60.0
         IF(DX(4).LT.-60.0) DX(4)=-60.0
         IF(X(8).GE.20.0.AND.DX(8).GT.0.0) DX(8)=0.0
         IF(X(8).LE.-23.0.AND.DX(8).LT.0.0) DX(8)=0.0
         IF(DX(8).GT.52.0) DX(8)=52.0
         IF(DX(8).LT.-52.0) DX(8)=-52.0
      END IF
      RETURN
      END
C
C END SUBROUTINE F1 -----------------------------------------------
```

```
C
C
      SUBROUTINE GCSTAR(X,NFLAG)
C
C*******************************************************************
C
C SUBROUTINE TO CALCULATE THE CONTROLS AT EACH SAMPLE TIME.
C ANTI-WINDUP COMPENSATED IF NFLAG=1.
C
C*******************************************************************
C
      REAL X(11),DEL(11),DEL2(11)
      INTEGER NFLAG
      COMMON/MATRIX/A(3,11),C(4,11),KX(2,11),KZ(2,4),KXM(2,4),
     1 KXU(2,4),PHI(4,4),PHINT(4,4),CM(4,4)
      COMMON/CONTRL/UNEW(2),UOLD(2),UCMD(4),UCOLD(4),XOLD(11),
     1 XMOLD(4),XM(4),NFLAG
      CALL MATML(PHI,XMOLD,XM,4,4,1)
      CALL MATML(PHINT,UCMD,DEL,4,4,1)
      CALL MATAD(XM,DEL,XM,4,1)
      CALL MATSB(X,XOLD,DEL,11,1)
      CALL MATML(KX,DEL,DEL2,2,11,1)
      CALL MATSB(UOLD,DEL2,UNEW,2,1)
      CALL MATSB(XM,XMOLD,DEL,4,1)
      CALL MATML(KXM,DEL,DEL2,2,4,1)
      CALL MATAD(UNEW,DEL2,UNEW,2,1)
      CALL MATSB(UCMD,UCOLD,DEL,4,1)
      CALL MATML(KXU,DEL,DEL2,2,4,1)
      CALL MATAD(UNEW,DEL2,UNEW,2,1)
      CALL MATML(CM,XMOLD,DEL,4,4,1)
      CALL MATML(C,XOLD,DEL2,4,11,1)
      CALL MATSB(DEL,DEL2,DEL,4,1)
      CALL MATML(KZ,DEL,DEL2,2,4,1)
      CALL MATAD(UNEW,DEL2,UNEW,2,1)
      IF(NFLAG.EQ.1) THEN
         IF(UNEW(1).GT.75.0-2.0*X(4)) UNEW(1)=75.0-2.0*X(4)
         IF(UNEW(1).LT.-75.0-2.0*X(4)) UNEW(1)=-75.0-2.0*X(4)
         IF(UNEW(2).GT.60.0-2.0*X(8)) UNEW(2)=60.0-2.0*X(8)
         IF(UNEW(2).LT.-69.0-2.0*X(8)) UNEW(2)=-69.0-2.0*X(8)
         IF(UNEW(1).GT.3.6+X(4)) UNEW(1)=3.6+X(4)
         IF(UNEW(1).LT.-3.6+X(4))  UNEW(1)=-3.6+X(4)
         IF(UNEW(2).GT.3.12+X(8)) UNEW(2)=3.12+X(8)
         IF(UNEW(2).LT.-3.12+X(8)) UNEW(2)=-3.12+X(8)
      END IF
      RETURN
      END
C
C END SUBROUTINE GCSTAR ---------------------------------------------
C
C
      SUBROUTINE RPOUT(A,M,N)
C
C*******************************************************************
C
```

```
C THIS ROUTINE PRINTS OUT A REAL MATRIX A
C
C*****************************************************************
C
      REAL A(M,N)
      INTEGER I,J,N,M
      DO 200 I=1,M
         PRINT'(" ",5(E11.4,3X))',(A(I,J),J=1,N)
         PRINT*
 200  CONTINUE
      END
C
C END SUBROUTINE RPOUT ------------------------------------------
C
C
      SUBROUTINE SETPLT(A,N,M,X)
C
C*****************************************************************
C
C THIS ROUTINE CONVERTS A REAL MATRIX OF DIMENSION N BY M INTO A
C VECTOR THAT IS COMPATIBLE WITH R.M. FLOYD'S PRINTER PLOTTING
C ROUTINE, PLOTLP.  THE INPUT MATRIX IS A.
C N= ROW DIMENSION OF A, THE NUMBER OF POINTS TO BE PLOTTED
C M= COLUMN DIMENSION OF A, THE NUMBER OF FUNCTIONS TO BE PLOTTED +1
C X= THE PLOTTING VECTOR, DIMENSION N*M
C
C*****************************************************************
C
      REAL A(N,M),X(N*M)
      INTEGER N,M,I,J
      DO 100 J=1,M
         DO 100 I=1,N
            X(I+(J-1)*N)=A(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE SETPLT -----------------------------------------
C
C
      SUBROUTINE PLOTLP(A,N,M,IPSC,ISCL,LPTERM,TITLE)
C
C*****************************************************************
C
C THIS ROUTINE WAS ADAPTED FROM R.M. FLOYD'S THESIS TO PRODUCE
C PRINTER PLOTS OF COMPUTED RESULTS.
C A= VECTOR OF DATA, CONVERTED FROM MATRIX FORM BY SUBROUTINE SETPLT
C N= NUMBER OF POINTS (INDEPENDENT VARIABLE) TO BE PLOTTED
C M= NUMBER OF FUNCTIONS (DEPENDENT VARIABLES) TO BE PLOTTED
C IPSC = -1-->ALL VARIABLES SCALED TOGETHER (1 PLOT)
C      =  0-->SCALED TOGETHER AND SEPARATELY (2 PLOTS)
C      = +1-->SCALED SEPARATELY (1 PLOT)
C ISCL =  0-->PLOT OVER EXACT RANGE OF VARIABLE
C        +1-->PLOT WITH EVEN SCALING
C LPTERM =  0-->PLOT 50 CHARACTERS WIDE
```

```
C         +1--)PLOT 100 CHARACTERS WIDE
C TITLE = MAX OF 50 CHARACTERS, TYPE CHARACTER
C
C***********************************************************************
C
      REAL YSCAL(6),YMIN(6),YPR(11),RISPAC,RMIN,RMAX,YL,YH,XPR,A(*)
      REAL SCAL
      INTEGER IBLNK(6),IPSC,ISCL,LPTERM,IPAPER,ISPAC,IPRTI,ISC,J,IC,IX
      INTEGER IL,JP,ITEMP,M1,M2,M,N,ICO,I
      CHARACTER TITLE*50
      CHARACTER*1 BLANK,PLUS,COLON,GRID,SYMBOL(6),OUT(101)
      DATA BLANK,PLUS,COLON,SYMBOL(1),SYMBOL(2)/' ','+',':','1','2'/
      DATA SYMBOL(3),SYMBOL(4),SYMBOL(5),SYMBOL(6)/'3','4','5','6'/
      IPAPER=5*(1+LPTERM)
      ISPAC=10*IPAPER
      RISPAC=REAL(ISPAC)
      ISPAC=ISPAC+1
      IPRTI=IPAPER+1
      RMIN=A(N+1)
      RMAX=RMIN
 25   DO 41 ISC=1,M
         M1=ISC*N+1
         YL=A(M1)
         YH=YL
         M2=N*(ISC+1)
         DO 40 J=M1,M2
            IF(A(J).LT.YL)THEN
               YL=A(J)
            END IF
            IF(A(J).GT.YH)THEN
               YH=A(J)
            END IF
 40      CONTINUE
         IF(YL.LT.RMIN)THEN
            RMIN=YL
         END IF
         IF(YH.GT.RMAX)THEN
            RMAX=YH
         END IF
         IF(IPSC.GE.0)THEN
            CALL VARSCL(YL,YH,YSCAL(ISC),RISPAC,ISCL)
         END IF
         YMIN(ISC)=YL
 41   CONTINUE
      IF(IPSC.LE.0) THEN
         CALL VARSCL(RMIN,RMAX,SCAL,RISPAC,ISCL)
      END IF
      IC=2-IABS(IPSC)
      DO 42 IX=1,ISPAC
         OUT(IX)=BLANK
 42   CONTINUE
      DO 100,ICO=1,IC
         PRINT'("1",11X,A50)',TITLE
         WRITE(4,'(11X,A50)')TITLE
      WRITE(4,'(A1)')BLANK
      PRINT*
      DO 60 I=1,N
         XPR=A(I)
         IF(MOD(I,10).EQ.0)THEN
            GRID=COLON
         ELSE
            GRID=BLANK
         END IF
         DO 44 IX=2,ISPAC,2
            OUT(IX)=GRID
   44    CONTINUE
         DO 46 IX=1,ISPAC,10
            OUT(IX)=PLUS
   46    CONTINUE
         DO 55 J=1,M
            IL=I+J*N
            IF(IPSC.EQ.-1)THEN
              JP=INT((A(IL)-RMIN)/SCAL)+1
            ELSE IF(IPSC.EQ.0)THEN
               IPSCT=IPSC+ICO
               IF(IPSCT.EQ.2)THEN
                  JP=INT((A(IL)-YMIN(J))/YSCAL(J))+1
               ELSE
                  JP=INT((A(IL)-RMIN)/SCAL)+1
               END IF
            ELSE
              JP=INT((A(IL)-YMIN(J))/YSCAL(J))+1
            END IF
   50       OUT(JP)=SYMBOL(J)
            IBLNK(J)=JP
   55    CONTINUE
         PRINT'(" ",F11.4,6X,101A1)',XPR,(OUT(IX),IX=1,ISPAC)
         WRITE(4,'(F11.4,6X,101A1)')XPR,(OUT(IX),IX=1,ISPAC)
         DO 59 J=1,M
            ITEMP=IBLNK(J)
            OUT(ITEMP)=BLANK
   59    CONTINUE
   60 CONTINUE
      IF(IPSC.NE.1)THEN
         IF(IPSCT.NE.2)THEN
            YPR(1)=RMIN
            DO 70 I=1,IPAPER
               YPR(I+1)=YPR(I)+10.*SCAL
   70       CONTINUE
            PRINT'("0    SCALE    ",11E10.3)',(YPR(I),I=1,IPRTI)
            WRITE(4,'(A1)')BLANK
            WRITE(4,'("     SCALE    ",11E10.3)')(YPR(I),I=1,IPRTI)
            WRITE(4,'(A1)')BLANK
            WRITE(4,'(A1)')BLANK
         END IF
      END IF
      IF(IPSC.EQ.1.OR.IPSCT.EQ.2)THEN
         DO 76 ISC=1,M
```

```
               YPR(1)=YMIN(ISC)
               DO 74 I=1,IPAPER
                  YPR(I+1)=YPR(I)+10.*YSCAL(ISC)
 74            CONTINUE
               PRINT'("0    SCALE ",A1,1X,11E10.3)',SYMBOL(ISC),(YPR(IX)
     1,IX=1,IPRTI)
               WRITE(4,'(A1)')BLANK
               WRITE(4,'("    SCALE ",A1,1X,11E10.3)')SYMBOL(ISC),
     1(YPR(IX),IX=1,IPRTI)
 76         CONTINUE
         END IF
         DO 90 ISC=1,56-N
            WRITE(4,'(A1)')BLANK
 90      CONTINUE
 100  CONTINUE
      PRINT'("1")'
      END
C
C END SUBROUTINE PLOTLP ---------------------------------------------
C
C
      SUBROUTINE VARSCL(XMIN,XMAX,SCALE,RSPACE,ISCL)
C
C*******************************************************************
C
C THIS IS A SCALING ROUTINE THAT SUPPORTS PLOTLP
C ADAPTED FROM R.M. FLOYD'S THESIS
C
C*******************************************************************
C
      REAL XMIN,XMAX,SCALE,RSPACE,EXP,XMINT,XMAXT
      INTEGER ISCL,ISCAL
      IF(XMAX.EQ.XMIN)THEN
         XMIN=.9*XMIN-10.
      END IF
      SCALE=XMAX-XMIN
      IF(ISCL.NE.0)THEN
         EXP=INT(100.+LOG10(SCALE))-100.
         FACTOR=10.**(1.-EXP)
         XMINT=XMIN*FACTOR
         XMAXT=XMAX*FACTOR
         IF(XMAXT.GE.0.)THEN
            XMAXT=XMAXT+.9
         END IF
         IF(XMINT.LE.0.)THEN
            XMINT=XMINT-.9
         END IF
         XMINT=AINT(XMINT)
         ISCAL=XMAXT-XMINT
         IF(MOD(ISCAL,5).NE.0)THEN
            ISCAL=ISCAL+5-MOD(ISCAL,5)
         END IF
         FACTOR=10.**(EXP-1.)
         XMIN=XMINT*FACTOR
         SCALE=FACTOR*REAL(ISCAL)
      END IF
      SCALE=SCALE/RSPACE
      END
C
C END SUBROUTINE VARSCL ----------------------------------------------
C
C
      SUBROUTINE EDIT(EDMAT,M,N)
C
C*********************************************************************
C
C THIS ROUTINE ALLOWS THE USER TO EDIT AN M BY N MATRIX EDMAT
C
C*********************************************************************
C     .
      REAL EL,EDMAT(M,N)
      INTEGER M,N,I,J
      CHARACTER ANSW*1
 10   PRINT*,'LIST CURRENT VALUES? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 10
      IF(ANSW.EQ.'Y') CALL RPOUT(EDMAT,M,N)
      PRINT*,'ENTER 0,0,0 <CR> WHEN ALL CHANGES HAVE BEEN MADE'
 100  PRINT*,'ENTER ROW #, COLUMN #, AND MATRIX ELEMENT: '
 110  READ*,I,J,EL
      IF(I.GT.0.AND.I.LE.M.AND.J.GT.0.AND.J.LE.N)THEN
         EDMAT(I,J)=EL
         GO TO 110
      ELSE IF(I.EQ.0.AND.J.EQ.0)THEN
 150  PRINT*,'LIST MODIFIED MATRIX? Y/N: '
      READ(*,'(A)')ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'N') GO TO 150
      IF(ANSW.EQ.'Y') CALL RPOUT(EDMAT,M,N)
 200     PRINT*,'ANY MORE CHANGES TO THIS MATRIX? Y/N: '
         READ(*,'(A)')ANSW
         IF(ANSW.NE.'Y'.AND.ANSW.NE.'N')GO TO 200
         IF(ANSW.EQ.'Y')GO TO 100
         IF(ANSW.EQ.'N')RETURN
      ELSE
         PRINT*,'SUBSCRIPT OUT OF RANGE'
         GO TO 100
      END IF
      END
C
C END SUBROUTINE EDIT-------------------------------------------------
C
C
      SUBROUTINE MATML(A,B,C,L,M,N)
C
C*********************************************************************
C
C THIS ROUTINE WILL MULTIPLY TWO REAL MATRICES
C A=AN L BY M MATRIX
```

```
C B=AN M BY N MATRIX
C C=THE L BY N PRODUCT OF A AND B
C NOTE: ACTUAL ARGUMENT C MUST DIFFER FROM A AND B
C
C************************************************************
C
      REAL A(L,M),B(M,N),C(L,N)
      INTEGER I,J,K,L,M,N
      DO 100 I=1,L
         DO 100 J=1,N
            C(I,J)=0.0
  100 CONTINUE
      DO 200 I=1,L
         DO 200 J=1,N
            DO 200 K=1,M
               C(I,J)=C(I,J)+A(I,K)*B(K,J)
  200 CONTINUE
      END
C
C END SUBROUTINE MATML ----------------------------------------
C
C
      SUBROUTINE MATAD(A,B,C,L,M)
C
C************************************************************
C
C THIS ROUTINE ADDS TWO REAL MATRICES OF DIMENSION L BY M
C A AND B ARE THE INPUTS, C IS THE SUM
C
C************************************************************
C
      REAL A(L,M),B(L,M),C(L,M)
      INTEGER I,J,L,M
      DO 100 I=1,L
         DO 100 J=1,M
            C(I,J)=A(I,J)+B(I,J)
  100 CONTINUE
      END
C
C END SUBROUTINE MATAD ----------------------------------------
C
C
      SUBROUTINE MATSB(A,B,C,L,M)
C
C************************************************************
C
C THIS ROUTINE SUBTRACTS REAL MATRIX B FROM REAL MATRIX A
C DIFFERENCE IS RETURNED IN REAL MATRIX C.
C ALL THREE MATRICES ARE OF DIMENSION L BY M
C
C************************************************************
C
      REAL A(L,M),B(L,M),C(L,M)
      INTEGER I,J,L,M
      DO 100 I=1,L
         DO 100 J=1,M
            C(I,J)=A(I,J)-B(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE MATSB --------------------------------------------
C
C
      SUBROUTINE SMUL(A,B,C,L,M)
C
C*******************************************************************
C
C THIS ROUTINE MULTIPLIES A REAL MATRIX BY A REAL SCALAR
C A= THE SCALAR
C B= THE MATRIX
C C= THE PRODUCT
C B AND C ARE OF DIMENSION L BY M
C
C*******************************************************************
C
      REAL A,B(L,M),C(L,M)
      INTEGER I,J,L,M
      DO 100 I=1,L
         DO 100 J=1,M
            C(I,J)=A*B(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE SMUL ----------------------------------------------
C
C
      SUBROUTINE COPYMT(A,B,N,M)
C
C*******************************************************************
C
C THIS ROUTINE COPIES A REAL MATRIX A INTO REAL MATRIX B.
C BOTH MATRICES ARE OF DIMENSION N BY M.
C
C*******************************************************************
C
      REAL A(N,M),B(N,M)
      INTEGER I,J,N,M
      DO 100 I=1,N
         DO 100 J=1,M
            B(I,J)=A(I,J)
 100  CONTINUE
      END
C
C END SUBROUTINE COPYMT --------------------------------------------
C
C
      SUBROUTINE DSCRT(A,N,TSAMP,PHI,PHINT,M,TP,TIDENT,CWORK)
C
```

```
C************************************************************************
C
C THIS ROUTINE APPROXIMATES THE STATE TRANSITION MATRIX AND ITS
C INTEGRAL FOR A TIME INVARIANT LINEAR SYSTEM AS A MATRIX EXPONENTIAL
C OVER A SMALL SAMPLE PERIOD.  RESULTS RETURNED IN REAL MATRICES.
C A= SYSTEM DYNAMICS MATRIX, TYPE REAL
C N= STATE DIMENSION
C TSAMP= SAMPLING PERIOD
C PHI= STATE TRANSITION MATRIX, TYPE REAL
C PHINT= APPROXIMATE INTEGRAL OF PHI, TYPE REAL
C M= NUMBER OF TERMS USED IN EXPONENTIAL EXPANSION
C TP, TIDENT AND CWORK ARE DUMMY ARRAYS
C
C************************************************************************
C
      REAL A(N,N),PHINT(N,N),PHI(N,N),TIDENT(N,N),TP(N,N)
      REAL CWORK(N,N)
      REAL TSAMP,RIJ
      INTEGER I,J,M,N
      DO 200 I=1,N
         DO 100 J=1,N
            TIDENT(I,J)=0.0
 100     CONTINUE
         TIDENT(I,I)=1.0
 200  CONTINUE
      CALL SMUL(TSAMP,TIDENT,PHINT,N,N)
      CALL COPYMT(PHINT,TP,N,N)
      CALL SMUL(TSAMP,A,PHI,N,N)
      DO 300 I=1,M
         CALL MATML(TP,PHI,CWORK,N,N,N)
         CALL COPYMT(CWORK,TP,N,N)
         RIJ=1.0/REAL(I+1)
         CALL SMUL(RIJ,TP,TP,N,N)
         CALL MATAD(PHINT,TP,PHINT,N,N)
 300  CONTINUE
      CALL MATML(A,PHINT,TP,N,N,N)
      CALL MATAD(TIDENT,TP,PHI,N,N)
C
C END SUBROUTINE DSCRT -----------------------------------------
C
      END
```

D. Modified CGT/PI/KF Design Software

D.1 Introduction

This appendix discusses the CGT/PI/KF design software that was used in this study. The program was originally written by Capt R. M. Floyd [16]. It was modified by Lt A. Moseley [34] to provide an interface with additional performance evaluation software and to incorporate the ability to use implicit model-following. Further modifications were made prior to and during this study to enhance the implicit model-following design capability, and to incorporate a minor correction to the original code that was suggested by Capt Floyd [17]. The resulting version has been referred to in this thesis as CGTPIV. Thc need for an additional correction to the code was discovered after the design work of this study had been completed [17]. Appendix E fully documents that error, how it was corrected, and its impact on the results of this study. That correction has been added, for completeness, to the source listing in Section D.4.

Extensive programming and usage guides for the earlier Floyd and Moseley versions of the program were provided in [16,34]: specifically, Volume 2 of each. Since familiarity with these works is really a prerequisite for intelligent use of the software, the information therein is not repeated here. Only changes made for this study are covered in this appendix.

D.2 Program Changes

All of the changes made to the original program written by Capt Floyd [16] are annotated on the source listing in Section D.4. The first correction suggested by Capt Floyd [17] is enclosed in a dashed-line box, and includes lines 9970 through 10010. The correction deals with choice of matrix manipulation routines which could, in some circumstances, affect the correct development of the cross-weighting terms of the discrete-time state weighting matrix for the regulator. The second correction suggested by Capt Floyd is similarly annotated at lines 10435 and 10440, and is explained in Appendix E. The rest of the dashed-line boxes throughout the listing are changes made by Lt Moseley, not all of which were documented in his thesis [34]. The section of code starting at line 27050, and proceeding through the end of the listing, was added by Lt Moseley.

The rest of the changes to the original code were made as a result of this thesis effort, and are annotated on the listing by solid-line boxes. In the version of the program developed by Lt Moseley, implicit model-following could only be used in regulator design immediately after the step during which a CGT was designed. There was apparently no reason for this other than expediency in ensuring that implicit model-following was not attempted without a command model having been established. The changes marked at line 3810, 3930-4080, 4860, 4890, 6450, 6580, 8790, 9060, 9320, 9460-9480, 10970 and 28170, as well as the deletions in subroutines "SCMD" and "IMPLEX" were all made to remove this restriction. The "IMPLIC" flag is initialized in line 3810 to indicate that implicit model-following has not yet been selected for the current design run. Once the regulator design option is selected, the code at line 3930 offers the implicit model-following option. Acceptance of the option sets "IMPLIC" to 1 and branches to subroutine "SETUP" to allow the definition or redefinition of the command model for use as the implicit regulator command model. The changes to "SETUP" at lines 4860 and 4890 allow for a value of 4 for the flag "ITYPE", which is used to tell subroutine "SCMD" that changes to the command model are to be made, but that a CGT design is not being pursued, as shown at lines 6450 and 6580. Initialization of the "NEWCM" flag, for which a value of 1 signifies a change to the command model, has been moved from SCMD to line 10970 to ensure that subsequent CGT designs implement the most recently defined command model. The changes at lines 8790, 9060 and 9320 add "IMPLIC" as a calling argument to the regulator design subroutines. The changes at line 9460 ensure that "IMPLEX" is only called when implicit model-following is being pursued, since the function that it would have performed under other circumstances is now being performed at line 3930; for the same reason, "IMPLIC" has been removed as a calling argument at line 28170, and the previously mentioned deletions to "IMPLEX" made.

The changes in subroutine "RSYS" (changes at lines 21520 and 21690-21700 as well as the deletion) correct a rather serious error in the implicit model-following code. Without the corrections, a change to the command model dynamics matrix or design model control matrix could only affect the variables used to calculate the implicit regulator weighting matrices, $\underline{\hat{Q}}_I$,$\underline{\hat{S}}_I$ and $\underline{\hat{R}}_I$ of (A-45), if the model that was changed had not previously been written to the SAVE file. In CGTPIV, all model dynamics and control matrices are discretized immediately after they are defined, and the continuous-time matrices are overwritten to save storage space. Development of the implicit state weighting matrix through (A-45) requires the use of the command model dynamics matrix and the continuous-time design model control matrix. When the program was modified to incorporate implicit model-following, storage for these matrices was allocated, but the program steps which altered the values in the arrays after model redefinition were bypassed if the model that had been redefined had already been written to the SAVE file during the current program run. When the required steps were bypassed, no indication was given the user that the changes had not had an effect on the regulator design, other than getting the wrong answer.

The changes at lines 28760-28870 simply allow the user to review the $\hat{\underline{Q}}_I$ matrix after the implicit quadratic weights ($\underline{Q}_I$ and $\underline{R}_I$ matrices) have been entered. In case the distribution of state weights is not what the designer wants, the option to change the $\underline{Q}_I$ and $\underline{R}_I$ matrices is offered prior to completing the regulator design.

An interactively executable load module for CGTPIV can be achieved using the segmentation job control file listed in Section D.5. With the changes made for this study, the program requires just over 65,000 octal words of memory for execution.

D.3 Using the Modified Software

Preparation for executing CGTPIV is exactly as outlined in the instructions of [16,34]. The only change as far as the user is concerned is that more design flexibility is available during execution. The modified code allows the user to employ implicit model-following for all regulator designs without requiring that a CGT design be conducted. An implicit model-following regulator can thus be designed by iteratively changing either the quadratic weightings or the regulator command model, or both. There is still only one "command model" in memory at any given time. During the CGT design and evaluation process, it functions as a CGT command model. During regulator designs, it functions as a regulator command model. Since the regulator design model is only used to define the constant gains for the regulator, the functions are distinct. At each entry into the CGT or implicit model-following regulator design path, the option is offered to change the command model, so the models used for the two design functions need not be the same.

A sample execution of the program follows. The load module is the file CGTPIV. The DATA file contains the design model for the AFTI F-16 used in this study, and a two-state command model, as in (V-4) through (V-6), in which P = 5. The sample is short, showing only the ways in which CGTPIV differs from previous versions. The software correction in lines 10435 and 10440, documented in Appendix E, was incorporated prior to the sample run, so the results are correct.

```
COMMAND- attach,d52dat

 PFN IS
 D52DAT
 AT CY= 001 SN=AFIT

COMMAND- copy,d52dat,data

COMMAND- attach,cgtpiv

 PFN IS
 CGTPIV
 AT CY= 001 SN=AFIT

COMMAND- cgtpiv

                          * * * CGTPIF * * *
                PROGRAM TO DESIGN A COMMAND GENERATOR TRACKER
          USING A REGULATOR WITH PROPORTIONAL PLUS INTEGRAL CONTROL
                  AND A KALMAN FILTER FOR STATE ESTIMATION.
                          * * * CGTPIF * * *

          DATE :  11/02/83

          TIME :  16.59.57.


ENTER SAMPLE PERIOD FOR DIGITAL CONTROLLER >.02

READ DESIGN  MODEL FROM 'DATA' FILE (Y OR N) >y

MODIFY MATRIX ELEMENTS (Y OR N) >n

WRITE DESIGN  MODEL TO 'SAVE' FILE (Y OR N) >n

POLES OF DESIGN  MATRIX

       1.3466988E-03  +J(  0.            )
       1.2970761E+00  +J(  0.            )
      -3.6664228E+00  +J(  0.            )
      -2.0000000E+01  +J(  0.            )
      -2.0000000E+01  +J(  0.            )

CONTROLLER DESIGN (Y OR N) >y

DESIGN REG/PI (Y OR N) >y

INCORPORATE IMPLICIT MODEL (Y OR N) >y

READ COMMAND MODEL FROM 'DATA' FILE (Y OR N) >y

MODIFY MATRIX ELEMENTS (Y OR N) >n
```

```
WRITE COMMAND MODEL TO 'SAVE' FILE (Y OR N) >n

POLES OF COMMAND MATRIX

     -5.0000000E+00  +J(  0.           )
     -5.0000000E+00  +J(  0.           )

ENTER WEIGHTS ON IMPLICIT OUTPUT DERIVATIVES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >1,1
2,1
0/

ENTER WEIGHTS ON IMPLICIT CONTROL MAGNITUDES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >1,1
2,1
0/

LIST QIH MATRIX TO TERMINAL (Y OR N) >y

QIH MATRIX

   50.01       -16.50       5.030        .8952        1.475
  -16.50        10.89      -1.9800E-02  -.5907       -.9735
   5.030      -1.9800E-02   1.000       1.0740E-03   1.7700E-03
   .8952       -.5907       1.0740E-03  3.2041E-02   5.2805E-02
   1.475       -.9735       1.7700E-03  5.2805E-02   8.7025E-02

CHANGE IMPLICIT WEIGHTS (Y OR N) >n

QI  MATRIX

   1.000
   1.000

RI  MATRIX

   1.000
   1.000

ENTER WEIGHTS ON OUTPUT DEVIATIONS:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >0/

ENTER WEIGHTS ON CONTROL MAGNITUDES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >0/

ENTER WEIGHTS ON CONTROL RATES:  2
ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >1,10
2,10
0/
```

```
Y   MATRIX

   0.
   0.

UM  MATRIX

   0.
   0.

MODIFY ELEMENTS OF 'X' MATRIX (Y OR N) >n

UR  MATRIX

    10.00
    10.00

KX  MATRIX

   -34.85        14.54       -1.956        1.429        .1587
    53.58       -57.87        .1482        .1515        .7372

KZ  MATRIX

   -.8226       -.7776
   -2.047        2.221

CONTROLLER EVALUATION WRT TRUTH MODEL (Y OR N) >n

POLES OF REGPI   MATRIX

      -1.2512981E+01  +J(  1.2946895E+01)
      -1.2512981E+01  +J( -1.2946895E+01)
      -2.6646129E+00  +J(  0.            )
      -5.0448375E+00  +J(  0.            )
      -2.5699700E+01  +J(  0.            )
      -1.8073609E+01  +J(  6.6542029E-01)
      -1.8073609E+01  +J( -6.6542029E-01)

ENTER STATE AND IC VALUE (0/ TERMINATES):  5 >0/

2 PLOTS OF 5 VARIABLES MAY BE PRINTED AT THE TERMINAL -- SPECIFY NUMBER
FOR EACH (N1,N2) >0/

MORE TIME RESPONSE RUNS (Y OR N) >n

CONTROLLER DESIGN (Y OR N) >n
```

```
FILTER DESIGN (Y OR N) >n

END DESIGN RUNS (Y OR N) >y

     REG/PI GAINS WRITTEN TO 'SAVE' FILE

PROGRAM EXECUTION STOP
     STOP
     065100 MAXIMUM EXECUTION FL.
      0.791 CP SECONDS EXECUTION TIME.
```

D.4 CGTPIV Source Listing

```
      PROGRAM MAIN(INPUT=64,OUTPUT=64,LIST=64,
     1 SAVE=64,DATA=64,PLOT=64,
     1 TAPE5=INPUT,TAPE6=OUTPUT,TAPE25=SAVE,TAPE50=DATA,
     2 TAPE99=PLOT,TAPE16=LIST)
C
C*********************************************************************
C
C CGT/PI/KF DESIGN PROGRAM (CGTPIV).  ORIGINAL VERSION WRITTEN BY CAPT
C R.M. FLOYD (1981).  MAJOR MODIFICATION MADE BY LT A. MOSELEY (1982).
C PREVIOUS VERSIONS ARE DOCUMENTED IN THE AUTHORS' THESES [16,34].
C
C DATE LAST REVISED: 28 OCT 83
C LIBRARIES USED: DLKLIB, ID=T820303
C
C*********************************************************************
C
      COMMON/MAIN1/NDIM,NDIM1,COM1(400)
      COMMON/MAIN2/COM2(400)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(2125)
      COMMON/ZMTX1/NVZM,ZM1(1225)
      COMMON/ZMTX2/ZM2(1225)
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1750)
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(225)
      COMMON/TRUMTX/NVTM,TM(1725)
      COMMON/CONTROL/NVCTL,CTL(900)
      COMMON/CREGPI/NVRPI,RPI(575)
      COMMON/CCGT/NVCGT,CGT(400)
      COMMON/CKF/NVFLT,FLT(690)
      COMMON/AMC/AM(100)
      COMMON/BDG/BD(75)
      NDIM=400
      NVSM=2125
      NVZM=1225
      NVDM=1750
      NVCM=225
      NVTM=1725
      NVCTL=900
      NVRPI=575
      NVCGT=400
      NVFLT=690
      KIN=5
      KSAVE=25
      KDATA=50
      KPLOT=99
      KLIST=16
      KTERM=6
      CALL CGTIO
      STOP
C END MAIN
      END
      SUBROUTINE DSND(ND)
      DIMENSION ND(1)
```

```
640=      ND(1)=8
650=      ND(2)=2
660=      ND(3)=2
670=      ND(4)=3
680=      ND(5)=0
690=      ND(6)=1
700=      ND(7)=0
710=      RETURN
720=C END SUBROUTINE DSND
730=      END
740=      SUBROUTINE DSNN(A,B,EX,G,Q,C,DY,EY,H,HN,R,AN,GN,QN)
750=      DIMENSION A(8,8),B(8,2),C(2,8),G(8),DY(2,2),H(3,8),R(3,3)
760=      DATA GRAVTY,DEGTRD,PI/32.174,.01745329,3.1415927/
770=      CALL ACDATA(LEVEL,VT,ALT,ALPHA,ZA,ZAD,ZQ,ZU,ZDE,ZDF,
780=     1 PMA,PMAD,PMQ,PMU,PMDE,PMDF,XA,XAD,XQ,XU,XDE,XDF,
790=     2 TE,DLX,BSPAN)
800= 10   ALPHAR=DEGTRD*ALPHA
810=      U0=VT*COS(ALPHAR)
820=      W0=VT*SIN(ALPHAR)
830=      A(1,3)=1.
840=      A(2,1)=-GRAVTY*SIN(ALPHAR)/U0
850=      A(2,2)=ZA
860=      A(2,3)=1.+ZQ
870=      A(3,2)=PMA
880=      A(3,3)=PMQ
890=      A(2,7)=ZA
900=      A(2,8)=ZQ
910=      A(3,7)=PMA
920=      A(3,8)=PMQ
930=      A(2,4)=ZDE
940=      A(2,5)=ZDF
950=      A(3,4)=PMDE
960=      A(3,5)=PMDF
970=      A(4,4)=-TE
980=      A(5,5)=-TE
990=      B(4,1)=TE
1000=     B(5,2)=TE
1010=     CALL GUSTS(LEVEL,ALT,SLU,SLW,SIGU,SIGW)
1020=     A(6,6)=-VT/SLW
1030=     A(7,6)=(1.-SQRT(3.))*SIGW*SQRT(-A(6,6))/SLW
1040=     A(7,7)=A(6,6)
1050=     A(8,8)=-VT*PI/4./BSPAN
1060=     A(8,6)=-A(8,8)*A(7,6)
1070=     A(8,7)=-A(8,8)*A(7,7)
1080=     G(6)=1.
1090=     G(7)=SIGW*SQRT(3.*VT/SLW)/VT
1100=     G(8)=-A(8,8)*G(7)
1110=     Q=1.
1120=     C(1,1)=1.
1130=     C(2,1)=1.
1140=     C(2,2)=-1.
1150=     H(1,1)=1.
1160=     H(2,2)=1.
1170=     H(3,3)=1.
```

```
      H(2,7)=1.
      R(1,1)=4.76E-6
      R(2,2)=1.22E-5
      R(3,3)=3.22E-5
      RETURN
C END SUBROUTINE DSNM
      END
      SUBROUTINE TRTHD(ND)
      DIMENSION ND(1)
      ND(1)=9
      ND(2)=2
      ND(3)=3
      ND(4)=1
      RETURN
C END SUBROUTINE TRTHD
      END
      SUBROUTINE TRTHM(AT,BT,GT,QT,HT,RT,TDT,TNT)
      DIMENSION AT(9,9),BT(9,2),GT(9),HT(3,9),RT(3,3),TDT(8,9)
      DATA GRAVTY,DEGTRD,PI/32.174,.01745329,3.1415927/
      CALL ACDATA(LEVEL,VT,ALT,ALPHA,ZA,ZAD,ZQ,ZU,ZDE,ZDF,
     1 PMA,PMAD,PMQ,PMU,PMDE,PMDF,XA,XAD,XQ,XU,XDE,XDF,
     2 TE,DLX,BSPAN)
   10 ALPHAR=DEGTRD*ALPHA
      U0=VT*COS(ALPHAR)
      W0=VT*SIN(ALPHAR)
      RZAD=1./(1.-ZAD)
      AT(1,3)=1.
      AT(2,1)=-GRAVTY*SIN(ALPHAR)/U0
      AT(2,2)=ZA
      AT(2,3)=1.+ZQ
      AT(2,4)=ZU
      AT(3,2)=PMA
      AT(3,3)=PMQ
      AT(3,4)=PMU
      AT(4,1)=-GRAVTY*COS(ALPHAR)
      AT(4,2)=XA
      AT(4,3)=XQ-W0
      AT(4,4)=XU
      AT(2,5)=ZDE
      AT(2,6)=ZDF
      AT(3,5)=PMDE
      AT(3,6)=PMDF
      AT(4,5)=XDE
      AT(4,6)=XDF
      AT(5,5)=-TE
      AT(6,6)=-TE
      AT(2,8)=ZA
      AT(2,9)=ZQ
      AT(3,8)=PMA
      AT(3,9)=PMQ
      AT(4,8)=XA
      AT(4,9)=XQ
      CALL GUSTS(LEVEL,ALT,SLU,SLW,SIGU,SIGW)
      AT(7,7)=-VT/SLW
      AT(8,7)=(1.-SQRT(3.))*SIGW*SQRT(-AT(7,7))/SLW
      AT(8,8)=AT(7,7)
      AT(9,9)=-VT*PI/4./BSPAN
      AT(9,7)=-AT(9,9)*AT(8,7)
      AT(9,8)=-AT(9,9)*AT(8,8)
      GT(7)=1.
      GT(8)=SIGW*SQRT(3.*VT/SLW)/VT
      GT(9)=-AT(9,9)*GT(8)
      QT=1.
      DO 20 I=1,9
      AT(2,I)=AT(2,I)*RZAD
      AT(3,I)=AT(3,I)+PMAD*AT(2,I)
   20 AT(4,I)=AT(4,I)+XAD*AT(2,I)
      BT(5,1)=TE
      BT(6,2)=TE
      HT(1,1)=1.
      HT(2,2)=1.
      HT(3,3)=1.
      HT(2,8)=1.
      RT(1,1)=4.76E-6
      RT(2,2)=1.22E-5
      RT(3,3)=3.22E-5
      TDT(1,1)=1.
      TDT(2,2)=1.
      TDT(3,3)=1.
      TDT(4,5)=1.
      TDT(5,6)=1.
      TDT(6,7)=1.
      TDT(7,8)=1.
      TDT(8,9)=1.
      RETURN
C END SUBROUTINE TRTHM
      END
      SUBROUTINE ACDATA(LEVEL,VT,ALT,ALPHA,ZA,ZAD,ZQ,ZU,ZDE,ZDF,
     1 PMA,PMAD,PMQ,PMU,PMDE,PMDF,XA,XAD,XQ,XU,XDE,XDF,
     2 TE,DLX,BSPAN)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DATA NENTRY/1/
    5 WRITE 101
      READ*,LEVEL
      IF((LEVEL.GT.3).OR.(LEVEL.LT.1)) GO TO 5
      WRITE 102
      READ*,VT,ALT,ALPHA
      WRITE 103
      READ*,ZA,ZAD,ZQ,ZU,ZDE,ZDF
      WRITE 104
      READ*,PMA,PMAD,PMQ,PMU,PMDE,PMDF
      WRITE 105
      READ*,XA,XAD,XQ,XU,XDE,XDF
      WRITE(KLIST,101)
      WRITE(KLIST,109) LEVEL
      WRITE(KLIST,102)
      WRITE(KLIST,110) VT,ALT,ALPHA
      WRITE(KLIST,103)
      WRITE(KLIST,110) ZA,ZAD,ZQ,ZU,ZDE,ZDF
      WRITE(KLIST,104)
      WRITE(KLIST,110) PMA,PMAD,PMQ,PMU,PMDE,PMDF
      WRITE(KLIST,105)
      WRITE(KLIST,110) XA,XAD,XQ,XU,XDE,XDF
      IF(NENTRY.EQ.0) GO TO 10
      BSPAN=30.
      DLX=13.798
      TE=20.
      RETURN
 10   WRITE 106
      READ*,TE
      WRITE 107
      READ*,DLX
      WRITE 108
      READ*,BSPAN
 101  FORMAT(" ENTER TURBULENCE LEVEL (1,2,3) >")
 102  FORMAT(" ENTER TRIM VELOCITY, ALTITUDE, AND ALPHA >")
 103  FORMAT(" ENTER ZA, ZAD, ZQ, ZU, ZDE, ZDF >")
 104  FORMAT(" ENTER MA, MAD, MQ, MU, MDE, MDF >")
 105  FORMAT(" ENTER XA, XAD, XQ, XU, XDE, XDF >")
 106  FORMAT(" ENTER TIME CONSTANT FOR ELEVATOR >")
 107  FORMAT(" ENTER DISTANCE FROM CG TO ACCELEROMETER >")
 108  FORMAT(" ENTER WING SPAN >")
 109  FORMAT(6X,I1)
 110  FORMAT(6(6X1PE15.7))
      RETURN
C END SUBROUTINE ACDATA
      END
      SUBROUTINE GUSTS(LEVEL,ALT,SLU,SLW,SIGU,SIGW)
      DIMENSION ATRB1(4),ATRB2(4),ATRB3(4),SIGT1(4),SIGT2(4),SIGT3(4)
      DATA ATRB1/2000.,2750.,10000.,30000./
      DATA ATRB2/2000.,2750.,10000.,45000./
      DATA ATRB3/2000.,5000.,20000.,70000./
      DATA SIGT1/4.5,5.,5.,0./
      DATA SIGT2/8.5,10.,10.,0./
      DATA SIGT3/12.,21.,21.,0./
      DATA IT1,IT2,IT3/1,1,1/
      IF(ALT-1750.) 5,15,15
 5    IF(ALT-1000.) 8,10,10
 8    ALTT=ALT
      GO TO 12
 10   ALTT=1000.
 12   SIGW=2.5*FLOAT(LEVEL)
      SIGU=1./(.177+8.23E-4*ALTT)**.4
      SLW=ALTT
      SLU=ALTT*SIGU**3
      SIGU=SIGU*SIGW
      GO TO 100
 15   SLU=1750.
      SLW=1750.
      IF(LEVEL-2) 17,18,16
 16   CALL TBLUP1(ATRB3,SIGT3,4,IT3,ALT,SIGU)
      GO TO 19
 17   CALL TBLUP1(ATRB1,SIGT1,4,IT1,ALT,SIGU)
      GO TO 19
 18   CALL TBLUP1(ATRB2,SIGT2,4,IT2,ALT,SIGU)
 19   SIGW=SIGU
 100  RETURN
C END SUBROUTINE GUSTS
      END
      SUBROUTINE TBLUP1(X,Y,N,IXP,XP,YP)
      DIMENSION X(1),Y(1)
      IF(IXP) 15,15,1
 1    IF(IXP-N) 10,10,5
 5    IXP=N
      GO TO 18
 10   IF(XP-X(IXP)) 12,18,20
 12   IXP=IXP-1
      IF(IXP) 15,15,10
 15   IXP=1
 18   YP=Y(IXP)
      RETURN
 20   IF(IXP-N) 21,18,5
 21   IXPP1=IXP+1
 22   IF(XP-X(IXPP1)) 25,30,30
 25   YP=Y(IXP)+(XP-X(IXP))/(X(IXPP1)-X(IXP))*(Y(IXPP1)-Y(IXP))
      RETURN
 30   IXP=IXPP1
      GO TO 20
C END SUBROUTINE TBLUP1
      END
      SUBROUTINE CGTXQ
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/MAIN2/COM2(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,LKXA12,LKXA13
      COMMON/CCGT/NVCGT,CGT(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      COMMON/AMC/AM(1)
      COMMON/BDG/BD(1)
      DIMENSION LD(15),ND(15)
      DATA NPLTZM/606/
      DATA IEOI,NO/-1,1HN/
      REWIND KLIST
      WRITE(KLIST,115) DATE(DUM),TIME(DUM)
      WRITE(KTERM,115) DATE(DUM),TIME(DUM)
  115 FORMAT("1",27X,"* * * CGTPIF * * *"/14X,
     1 "PROGRAM TO DESIGN A COMMAND GENERATOR TRACKER"/8X,
     2 "USING A REGULATOR WITH PROPORTIONAL PLUS INTEGRAL CONTROL"/16X,
     3 "AND A KALMAN FILTER FOR STATE ESTIMATION."/28X,
     4 "* * * CGTPIF * * *"//11X,"DATE : ",A10//,11X,
     5 "TIME : ",A10////)
      REWIND KSAVE
      REWIND KDATA
      WRITE(KSAVE,112) IEOI,NPLTZM
      DO 10 I=1,15
   10 ND(I)=0
      DO 12 I=1,15
   12 LD(I)=1
      LFLRPI=0
      LFLCGT=0
      LFLKF=0
      LTEVAL=0
      LABORT=0
      IPI=0
      ICGT=0
      ITRU=0
      IFLTR=0
      ICODE=4
      LFAVAL=0
      LGCGT=0
      NVCON=MIN0(NDIM,NVZM)
      KOUT=KLIST
      KPUNCH=KPLOT
      IF(NVSM.GE.NPLTZM) GO TO 50
      WRITE 101,NPLTZM
      GO TO 1000
   50 WRITE 102
      READ*,TSAMP
      IF(TSAMP.LE.0.) GO TO 50
      WRITE(KLIST,103) TSAMP
  103 FORMAT("0SAMPLE PERIOD IS ",F5.3," SECONDS")
      CALL SETUP(ND,LD,ICGT,ITRU,1)
      IF(LABORT) 1000,100,1000
  100 LABORT=0
      IMPLIC=0
      WRITE 104
  104 FORMAT("0CONTROLLER DESIGN (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 500
      LFLKF=0
      CALL PINTX(IPI)
      IF(LABORT) 1000,125,1000
  125 WRITE 105
  105 FORMAT("0DESIGN REG/PI (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 150
      WRITE 400
  400 FORMAT("0INCORPORATE IMPLICIT MODEL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 490
      IMPLIC=1
      CALL SETUP(ND,LD,ICGT,ITRU,4)
      IF(ICGT.NE.0) GO TO 460
      IMPLIC=0
      GO TO 480
  460 IF(NPD.EQ.NNC) GO TO 480
      WRITE 470
  470 FORMAT("0COMMAND MODEL STATE DIMENSION MUST EQUAL SYSTEM
     1OUTPUT DIMENSION")
      LABORT=-1
  480 IF(LABORT) 100,490,1000
  490 CALL SREGPI(IMPLIC)
      IF(LABORT) 1000,200,1000
  150 WRITE 106
  106 FORMAT("0DESIGN CGT (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 100
      CALL SETUP(ND,LD,ICGT,ITRU,2)
      IF(ICGT) 155,100,155
  155 IF(LABORT) 100,160,1000
  160 CALL SCGT
      IF(LABORT) 100,170,1000
  170 IF(LFLCGT.LE.0) GO TO 125
  200 LABORT=0
      WRITE 107
  107 FORMAT("0CONTROLLER EVALUATION WRT TRUTH MODEL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 250
      CALL SETUP(ND,LD,ICGT,ITRU,3)
      IF(LABORT) 200,260,1000
  250 LTEVAL=0
  260 CALL CEVAL
      IF(LFLCGT.EQ.1) LGCGT=1
      IF(LFAVAL.EQ.0.OR.LGCGT.EQ.0) GO TO 100
  270 WRITE 600
  600 FORMAT("0WRITE PERFORMANCE EVALUATION DATA TO 'SAVE' FILE (Y OR N)
     +>")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 100
      ICODE=ICODE+1
      CALL PFDATA(ICODE,ND)
      INUM=ICODE-4
      WRITE 605,INUM
  605 FORMAT("0PERFORMANCE EVALUATION DATA, NO. "I2,",WRITTEN TO 'SAVE
     +' FILE")
```

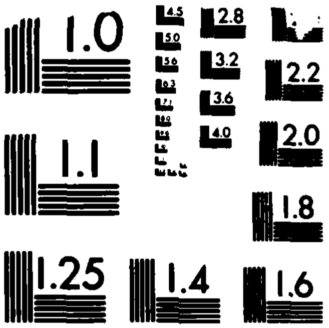
```
      GO TO 100
  500 LABORT=0
      WRITE 108
  108 FORMAT("0FILTER DESIGN (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 900
      CALL FLTRK(IFLTR)
      IF(IFLTR.EQ.0) GO TO 900
      IF(LABORT) 1000,510,1000
  510 CALL SETUP(ND,LD,ICGT,ITRU,3)
      IF(LABORT) 500,525,1000
  525 CALL FEVAL
  530  IF(LABORT) 1000,540,1000
  540  LFAVAL=1
       IF(LGCGT.EQ.1) GO TO 270
      GO TO 500
  900 WRITE 109
  109 FORMAT("0END DESIGN RUNS (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 100
      IF(LFLRPI.EQ.0) GO TO 1000
      NPNTS=NRD*NNPR
      ND(1)=NPNTS
      ND(2)=LKX
      ND(3)=LKZ
      CALL WFILED(4,NPNTS,ND,RPI(LKX))
      WRITE 113
 1000 CONTINUE
      WRITE(KLIST,110)
      REWIND KSAVE
      REWIND KDATA
      REWIND KLIST
      WRITE 110
  101 FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
  102 FORMAT("0ENTER SAMPLE PERIOD FOR DIGITAL CONTROLLER >")
  110 FORMAT("0PROGRAM EXECUTION STOP")
  111 FORMAT(A3)
  112 FORMAT(2I4)
  113 FORMAT(6X,"REG/PI GAINS WRITTEN TO 'SAVE' FILE")
      RETURN
C END SUBROUTINE CGTXQ
      END
      SUBROUTINE SETUP(ND,LD,ICGT,ITRU,ITYPE)
      DIMENSION ND(1),LD(1)
      GO TO (10,15,20,15) ITYPE
   10 CALL SDSN(ND,LD)
      RETURN
   15 CALL SCMD(ND,LD,ICGT,ITYPE)
      RETURN
   20 CALL STRTH(ND,LD,ITRU)
      RETURN
C END SUBROUTINE SETUP
      END
      SUBROUTINE SDSN(ND,LD)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      DIMENSION ND(1),LD(1)
      NSIZE=0
      CALL RSYS(SM,LD,ND,1,NSIZE)
      IF(LABORT.GT.0) RETURN
      NSIZE=NNPR
      IF(NPLD.GT.NSIZE) NSIZE=NPLD
      NSIZE=NSIZE*NSIZE
      IF(NSIZE.LE.NVCOM) GO TO 5
      WRITE 101,NSIZE
  101 FORMAT("0INSUFFICIENT MEMORY /MAIN1/,/MAIN2/,/ZMTX1/,/ZMTX2/, NEED
     1: ",I4)
      LABORT=NSIZE
      RETURN
    5 IF(NRD.EQ.NPD) GO TO 10
      WRITE 102
  102  FORMAT("0NUMBER OF INPUTS AND OUTPUTS MUST BE EQUAL FOR DESIGN")
      LABORT=-1
      RETURN
   10 CALL DSCRTD(LD,ZM1,ZM2)
      RETURN
C END SUBROUTINE SDSN
      END
      SUBROUTINE DSCRTD(LD,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      DIMENSION LD(1),ZM1(1),ZM2(1)
      NDIM=NPLD
      NDIM1=NDIM+1
      CALL POLES(SM,NND,1,ZM1,ZM2)
      DO 1 I=1,NND
    1 IF(ZM1(I).GT.0.) LFLCGT=-1
      CALL TFRMTX(SM,DM,NND,NND,2)
      LAP=1
      LGP=LAP+NPLD*NPLD
      IF(NWD.EQ.0) GO TO 5
      CALL TFRMTX(SM(LD(4)),DM(LGP),NND,NWD,2)
    5 IF(NDD.EQ.0) GO TO 10
      L1=LADDR(NPLD,NND+1,1)
      L2=LADDR(NPLD,1,NND+1)
      L3=LADDR(NPLD,NND+1,NND+1)
      CALL ZPART(DM(L1),NDD,NND,NPLD)
      CALL TFRMTX(SM(LD(3)),DM(L2),NND,NDD,2)
      CALL TFRMTX(SM(LD(12)),DM(L3),NDD,NDD,2)
      IF(NWD.EQ.0) GO TO 8
      L1=L1+LGP-1
      CALL ZPART(DM(L1),NDD,NWD,NPLD)
 8    L2=LADDR(NPLD,1,NWD+1)+LGP-1
      L3=LADDR(NPLD,NWD+1,NWD+1)+LGP-1
      CALL ZPART(DM(L2),NWD,NWDD,NPLD)
      CALL TFRMTX(SM(LD(13)),DM(L3),NDD,NWDD,2)
 10   LPHI=LGP+NPLD*NWPNWD
      LEADSN=1
      CALL NDSCRT(DM,NDIM,NT)
      CALL DSCRT(NPLD,DM,TSAMP,FLT,ZM1,NT)
      LFLTRK=LEADSN+NPLD*NPLD
      CALL TFRMTX(DM(LPHI),FLT,NWD,NWD,1)
      LBD=LPHI+NWD*NWD
      CALL TFRMTX(SM,ZM1,NWD,NWD,1)
      CALL FMMUL(SM,SM(LD(2)),NWD,NWD,NRD,DM(LBD))
      LEX=LBD+NWD*NRD
      IF(NDD.EQ.0) GO TO 15
      L1=LADDR(NPLD,1,NWD+1)
      CALL TFRMTX(DM(LEX),FLT(L1),NWD,NDD,1)
      LPHD=LEX+NWD*NDD
      L1=LADDR(NPLD,NWD+1,NWD+1)
      CALL TFRMTX(DM(LPHD),FLT(L1),NDD,NDD,1)
      LQ=LPHD+NDD*NDD
      GO TO 20
 15   LQ=LEX
 20   IF(NWD.EQ.0) GO TO 25
      CALL FTMTX(SM(LD(5)),DM(LQ),NWD,NWD)
      LQN=LQ+NWD*NWD
      GO TO 28
 25   LQN=LQ
 28   IF(NWDD.EQ.0) GO TO 33
      CALL FTMTX(SM(LD(14)),DM(LQN),NWDD,NWDD)
      LQD=LQN+NWDD*NWDD
      GO TO 35
 33   LQD=LQN
      IF(NWPNWD.GT.0) GO TO 35
      LC=LQD
      GO TO 36
 35   CALL QDSCRT(DM(LQ),DM(LQN),ZM1,ZM2)
      LC=LQD+NPLD*NPLD
 36   LDY=LC+NPD*NWD
      LEY=LDY+NPD*NRD
      LHP=LEY+NPD*NDD
      LR=LHP+NWD*NPLD
      L1=LR+NWD*NWD-LC
      CALL FTMTX(SM(LD(6)),DM(LC),L1,1)
      L1=LEY-1
      NODY=1
      DO 40 I=LDY,L1
      IF(DM(I).EQ.0.) GO TO 40
      NODY=0
      GO TO 45
 40   CONTINUE
 45   NOEY=1
      IF(NDD.LT.1) GO TO 55
      L1=LHP-1
      DO 50 I=LEY,L1
      IF(DM(I).EQ.0.) GO TO 50
      NOEY=0
      GO TO 55
 50   CONTINUE
 55   CALL MATLST(DM(LPHI),NND,NND,"PHI",KLIST)
      CALL MATLST(DM(LBD),NND,NRD,"BD",KLIST)
      IF(NWPNWD.GT.0) CALL MATLST(DM(LQD),NPLD,NPLD,"QD",KLIST)
      IF(NMD.GT.0) CALL MATLST(DM(LHP),NMD,NPLD,"HA",KLIST)
      IF(NDD.EQ.0) RETURN
      CALL MATLST(DM(LEX),NND,NDD,"EXD",KLIST)
      CALL MATLST(DM(LPHD),NDD,NDD,"PHN",KLIST)
      RETURN
C END SUBROUTINE DSCRTD
      END
      SUBROUTINE QDSCRT(Q,QN,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/NDIND/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      DIMENSION Q(1),QN(1),ZM1(1),ZM2(1)
      IF(NWD.EQ.0) GO TO 5
      CALL TFRMTX(Q,ZM1,NWD,NWD,2)
 5    IF(NWDD.EQ.0) GO TO 10
      L1=LADDR(NPLD,NWD+1,NWD+1)
      CALL TFRMTX(QN,ZM1(L1),NWDD,NWDD,2)
      IF(NWD.EQ.0) GO TO 10
      L1=LADDR(NPLD,1,NWD+1)
      CALL ZPART(ZM1(L1),NWD,NWDD,NPLD)
      L1=LADDR(NPLD,NWD+1,1)
      CALL ZPART(ZM1(L1),NWDD,NWD,NPLD)
 10   CALL MAT3(NPLD,NWPNWD,DM(LGP),ZM1,ZM2)
      CALL INTEG(NPLD,DM(LAP),ZM2,DM(LQD),TSAMP)
      RETURN
C END SUBROUTINE QDSCRT
      END
      SUBROUTINE SCMD(ND,LD,ICGT,ITYPE)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIND/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION ND(1),LD(1)
      DATA NO/1HN/
      IF(ITYPE.EQ.4) GO TO 10
      WRITE(KLIST,110)
  110 FORMAT(////11X,5("* "),"CGT DESIGN",5(" *")////)
/--------------deletion------------------/
      IF(LFLRPI) 10,5,10
    5 WRITE 102
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 8
      CALL READFS(SM,ND,4,IERR)
      NSIZE=ND(1)
      LKX=ND(2)
      LKZ=ND(3)
      CALL FTMTX(SM,RPI(LKX),NSIZE,1)
      IF(IERR.NE.0) RETURN
      CALL MATLST(RPI(LKX),NRD,NND,"KX",KLIST)
      CALL MATLST(RPI(LKZ),NRD,NRD,"KZ",KLIST)
      LFLRPI=-1
      GO TO 10
    8 IF(LFLCGT.GE.0) GO TO 9
      WRITE 103
  103 FORMAT("0SYSTEM UNSTABLE - - OPEN-LOOP CGT NOT FEASIBLE")
      RETURN
    9 LKX=1
      LKZ=1
      NSIZE=NRD*NND
      CALL ZPART(RPI(LKX),1,NSIZE,1)
   10 IF(ICGT.EQ.0) GO TO 12
      WRITE 108
  108 FORMAT(" MODIFY COMMAND MODEL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) RETURN
   12 CALL RSYS(SM,LD,ND,2,ICGT)
      IF(LABORT.NE.0) RETURN
      NEWCM=1
      CALL POLES(SM,NNC,2,ZM1,ZM2)
      IF(NPC.EQ.NPD) GO TO 15
      WRITE 104
      LABORT=-1
      RETURN
   15 CALL DSCRTC(LD,ZM1)
  102 FORMAT(" READ REG/PI GAINS FROM 'DATA' FILE (Y OR N) >")
  104 FORMAT("0COMMAND AND DESIGN MODEL OUTPUTS NOT EQUAL IN NUMBER")
  111 FORMAT(A3)
      RETURN
C END SUBROUTINE SCMD
      END
      SUBROUTINE DSCRTC(LD,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      DIMENSION LD(1),ZM1(1)
      NDIM=NNC
      NDIM1=NDIM+1
      CALL NDSCRT(SM,NDIM,NT)
      CALL DSCRT(NDIM,SM,TSAMP,CM,ZM1,NT)
      LPHC=1
      LBDC=LPHC+NNC*NNC
      CALL MMUL(ZM1,SM(LD(2)),NDIM,NDIM,NRC,CM(LBDC))
      LCC=LBDC+NNC*NRC
      LDC=LCC+NPC*NNC
      L1=LDC+NPC*NRC-LCC
      CALL FTMTX(SM(LD(3)),CM(LCC),L1,1)
      NODC=1
      L1=L1+LCC-1
      DO 10 I=LDC,L1
      IF(CM(I).EQ.0.) GO TO 10
      NODC=0
      GO TO 15
 10   CONTINUE
 15   CALL MATLST(CM,NNC,NNC,"PHM",KLIST)
      CALL MATLST(CM(LBDC),NNC,NRC,"BDM",KLIST)
      CALL MATLST(CM(LCC),NPC,NNC,"CM",KLIST)
      CALL MATLST(CM(LDC),NPC,NRC,"DM",KLIST)
      RETURN
C END SUBROUTINE DSCRTC
      END
      SUBROUTINE STRTH(ND,LD,ITRU)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      DIMENSION ND(1),LD(1)
      DATA NO/1HN/
      IF(ITRU.EQ.0) GO TO 5
      WRITE 103
 103  FORMAT(" MODIFY TRUTH MODEL (Y OR N) >")
      READ 111,IANS
 111  FORMAT(A3)
      IF(IANS.EQ.NO) GO TO 20
 5    CALL RSYS(SM,LD,ND,3,ITRU)
      IF(LABORT.GT.0) RETURN
      NSIZE=NNT*NNT
      IF(NSIZE.LE.NVCOM) GO TO 8
      WRITE 101,NSIZE
 101  FORMAT("0INSUFFICIENT MEMORY /MAIN1/,/MAIN2/,/ZMTX1/,/ZMTX2/, NEED
     1: ",I2)
      LABORT=NSIZE
      RETURN
 8    IF((NRT.EQ.NRD).AND.(NMT.EQ.NMD)) GO TO 10
      WRITE 102
 102  FORMAT("0INPUTS AND MEASUREMENTS MUST BE EQUAL IN NUMBER FOR DESIG
     1N AND TRUTH MODELS")
```

```
      LABORT=-1
      RETURN
 10   CALL POLES(SM,NNT,3,ZM1,ZM2)
      CALL DSCRTT(LD,ZM1)
 20   LTEVAL=1
      RETURN
C END SUBROUTINE STRTH
      END
      SUBROUTINE DSCRTT(LD,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIND/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDINT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      DIMENSION LD(1),ZM1(1)
      NDIM=NNT
      NDIM1=NDIM+1
      CALL NDSCRT(SM,NDIM,NT)
      CALL DSCRT(NDIM,SM,TSAMP,TM,ZM1,NT)
      LPHT=1
      LBDT=LPHT+NNT*NNT
      CALL MMUL(ZM1,SM(LD(2)),NDIM,NDIM,NRT,TM(LBDT))
      LQDT=LBDT+NNT*NRT
      IF(NWT.GT.0) GO TO 10
      LHT=LQDT
      GO TO 15
 10   CALL MAT3(NDIM,NWT,SM(LD(3)),SM(LD(4)),ZM1)
      CALL INTEG(NDIM,SM,ZM1,TM(LQDT),TSAMP)
      LHT=LQDT+NNT*NNT
 15   LRT=LHT+NMT*NNT
      LTDT=LRT+NMT*NMT
      LTNT=LTDT+NND*NNT
      L1=LTNT+NDD*NNT-LHT
      CALL FTMTX(SM(LD(5)),TM(LHT),L1,1)
      CALL MATLST(TM,NNT,NNT,"PHT",KLIST)
      CALL MATLST(TM(LBDT),NNT,NRT,"BDT",KLIST)
      IF(NWT.GT.0) CALL MATLST(TM(LQDT),NNT,NNT,"QDT",KLIST)
      RETURN
C END SUBROUTINE DSCRTT
      END
      SUBROUTINE PIMTX(IPI)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIND/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      IF(IPI.EQ.1) RETURN
      WRITE(KLIST,110)
  110 FORMAT(////11X,5("* "),"CONTROLLER SET-UP",5(" *")////)
      NDIM=NNPR
      NSIZE=NDIM*(2*NDIM+NPD)
      IF(NSIZE.LE.NVCTL) GO TO 10
      WRITE 101,NSIZE
  101 FORMAT("0INSUFFICIENT MEMORY /CONTROL/, NEED: ",I4)
      LABORT=NSIZE
      RETURN
   10 NDIM1=NDIM+1
      LPI11=1
      LPI12=LPI11+NND*NND
      LPI21=LPI12+NND*NRD
      LPI22=LPI21+NPD*NND
      LPHDL=LPI22+NPD*NRD
      CALL TFRMTX(DM(LPHI),ZM1,NND,NND,2)
      CALL SUBI(ZM1,NND,NDIM)
      L2=LADDR(NDIM,1,NND+1)
      CALL TFRMTX(DM(LBD),ZM1(L2),NND,NRD,2)
      L3=LADDR(NDIM,NND+1,1)
      CALL TFRMTX(DM(LC),ZM1(L3),NPD,NND,2)
      L4=LADDR(NDIM,NND+1,NND+1)
      CALL TFRMTX(DM(LDY),ZM1(L4),NPD,NRD,2)
      CALL GMINV(NDIM,NDIM,ZM1,ZM2,MR,1)
      IF(MR.EQ.NDIM) GO TO 15
      WRITE 102
      WRITE(KLIST,102)
  102 FORMAT("0PI MATRIX IS RANK DEFECTIVE")
   15 CALL MATLST(ZM2,NNPR,NNPR,"PI",KLIST)
      CALL TFRMTX(CTL(LPI11),ZM2,NND,NND,1)
      CALL TFRMTX(CTL(LPI12),ZM2(L2),NND,NRD,1)
      CALL TFRMTX(CTL(LPI21),ZM2(L3),NPD,NND,1)
      CALL TFRMTX(CTL(LPI22),ZM2(L4),NPD,NRD,1)
      CALL CDIF
      IPI=1
      RETURN
C END SUBROUTINE PIMTX
      END
      SUBROUTINE CDIF
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      CALL TFRMTX(DM(LPHI),CTL(LPHDL),NND,NND,2)
      L1=LADDR(NDIM,1,NND+1)+LPHDL-1
      CALL TFRMTX(DM(LBD),CTL(L1),NND,NRD,2)
      L1=LADDR(NDIM,NND+1,1)+LPHDL-1
      CALL ZPART(CTL(L1),NRD,NND,NDIM)
      L1=LADDR(NDIM,NND+1,NND+1)+LPHDL-1
      CALL IDNT(NRD,CTL(L1),1.)
      LBDL=LPHDL+NDIM*NDIM
      CALL ZPART(CTL(LBDL),NND,NRD,NDIM)
      L1=LADDR(NDIM,NND+1,1)+LBDL-1
      CALL IDNT(NRD,CTL(L1),1.)
      RETURN
C END SUBROUTINE CDIF
      END
      SUBROUTINE SREGPI(IMPLIC)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NMDD,NPLD,NWPNWD,NNPR
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      WRITE(KLIST,110)
  110 FORMAT(////11X,5("* "),"REG/PI DESIGN",5(" *")////)
      NSIZE=NRD*(4*NRD+NND)+NNPR*NNPR
      IF(NSIZE.LE.NVRPI) GO TO 5
      WRITE 101,NSIZE
  101 FORMAT("0INSUFFICIENT MEMORY /CREGPI/, NEED: ",I4)
      GO TO 8
    5 NSIZE=NNPR*(3*NNPR+NRD)
      IF(NSIZE.LE.NVSM) GO TO 10
      WRITE 102,NSIZE
  102 FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
    8 LABORT=NSIZE
      RETURN
   10 LX=1
      LU=LX+NNPR*NNPR
      CALL WXUS(SM(LX),SM(LU),COM1,ZM1,ZM2,IMPLIC)
      LUIST=LU+NNPR*NRD
      LPHP=LUIST+NNPR*NNPR
      CALL PXUP(CTL(LPHDL),CTL(LBDL),SM(LX),SM(LU),COM1,ZM2,
     1 SM(LUIST),SM(LPHP),SM(LX),ZM1)
      CALL DRIC(NDIM,SM(LPHP),ZM2,SM(LX),ZM1,RPI(LPHCL))
      CALL GCSTAR(SM(LPHP),CTL(LBDL),SM(LU),ZM1,SM(LUIST),SM(LX),ZM2)
      CALL TFRMTX(ZM1,SM(LX),NRD,NDIM,1)
      CALL FMMUL(ZM1,CTL(LPI11),NRD,NND,NND,RPI(LKX))
      L1=LADDR(NRD,1,NND+1)
      CALL FMMUL(ZM1(L1),CTL(LPI21),NRD,NRD,NND,ZM2)
      NDIM=NRD
      NDIM1=NDIM+1
      CALL MADD1(NRD,NND,RPI(LKX),ZM2,RPI(LKX),1.)
      CALL FMMUL(ZM1,CTL(LPI12),NRD,NND,NRD,RPI(LKZ))
      CALL FMMUL(ZM1(L1),CTL(LPI22),NRD,NRD,NRD,ZM2)
      CALL MADD1(NRD,NRD,RPI(LKZ),ZM2,RPI(LKZ),1.)
      CALL MATLST(RPI(LKX),NRD,NND,"KX",KLIST)
      CALL MATLST(RPI(LKX),NRD,NND,"KX",KTERM)
      CALL MATLST(RPI(LKZ),NRD,NRD,"KZ",KLIST)
      CALL MATLST(RPI(LKZ),NRD,NRD,"KZ",KTERM)
      LFLRPI=1
      LFLCGT=0
      RETURN
C END SUBROUTINE SREGPI
      END
      SUBROUTINE WXUS(X,U,S,ZM1,ZM2,IMPLIC)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION X(1),U(1),S(1),ZM1(1),ZM2(1)
      DATA NO/1HN/
      IF(IMPLIC.EQ.0) GO TO 2
      CALL IMPLEX(ZM1)
    2 IF(LFLRPI) 5,5,10
    5 LXDW=1
      LUDW=LXDW+2*NRD*NRD
      LPHCL=LUDW+NRD*NRD
      LKX=LPHCL+NNPR*NNPR
      LKZ=LKX+NRD*NND
      L1=LPHCL-1
      CALL ZPART(RPI,1,L1,1)
   10 LUX=NRD*NRD+1
      WRITE 101,NPD
  101 FORMAT(" ENTER WEIGHTS ON OUTPUT DEVIATIONS: ",I2)
      CALL ROWGTS(RPI,NPD,0)
      WRITE 102,NRD
  102 FORMAT(" ENTER WEIGHTS ON CONTROL MAGNITUDES: ",I2)
      CALL ROWGTS(RPI(LUX),NRD,1)
      WRITE 103,NRD
  103 FORMAT(" ENTER WEIGHTS ON CONTROL RATES: ",I2)
      CALL ROWGTS(RPI(LUDW),NRD,1)
      CALL MATLST(RPI,NPD,NPD,"Y",KLIST)
      CALL DVCTOR(NPD,RPI,ZM1)
      CALL MATLST(ZM1,NPD,1,"Y",KTERM)
      CALL DVCTOR(NRD,RPI(LUX),ZM1)
      CALL MATLST(ZM1,NRD,1,"UM",KTERM)
      CALL MATLST(RPI(LUX),NRD,NRD,"UM",KLIST)
      NDIM=NNPR
      NDIM1=NDIM+1
      CALL FORMX(RPI,RPI(LUX),DM(LC),DM(LDY),ZM2,ZM1,COM1)
      WRITE(KTERM,104)
  104 FORMAT("0MODIFY ELEMENTS OF 'X' MATRIX (Y OR N) >")
      READ 111,IANS
  111 FORMAT(A3)
      IF(IANS.EQ.NO) GO TO 20
      WRITE(KTERM,105)
  105 FORMAT(" LIST 'X' MATRIX TO TERMINAL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 12
      CALL MATLST(ZM2,NNPR,NNPR,"X",KTERM)
   12 CALL ZMATIN(ZM2,NNPR,NNPR,-1)
   20 CALL MATLST(ZM2,NNPR,NNPR,"X",KLIST)
      CALL MATLST(RPI(LUDW),NRD,NRD,"UR",KLIST)
      CALL DVCTOR(NRD,RPI(LUDW),ZM1)
      CALL MATLST(ZM1,NRD,1,"UR",KTERM)
      IF(IMPLIC.EQ.0) GO TO 260
         CALL MODIFX(ZM2)
  260 CONTINUE
      T1=.25*TSAMP
      CALL MSCALE(ZM1,ZM2,NDIM,NDIM,T1)
      CALL DIAG(NDIM,COM1,CTL(LPHDL),1.,1.)
      CALL MAT3A(NDIM,NDIM,COM1,ZM1,X)
      CALL MAT3A(NRD,NDIM,CTL(LBDL),ZM1,U)
      CALL MAT4A(COM1,ZM1,NDIM,NDIM,NDIM,ZM2)
      CALL MMUL(ZM2,CTL(LBDL),NDIM,NDIM,NRD,S)
      CALL TFRMTX(RPI(LUDW),ZM1,NRD,NRD,2)
      CALL MADD1(NRD,NRD,U,ZM1,U,TSAMP)
      RETURN
C END SUBROUTINE WXUS
      END
      SUBROUTINE FORMX(QY,RY,C,D,X,Z1,Z2)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      DIMENSION QY(1),RY(1),C(1),D(1),X(1),Z1(1),Z2(1)
      CALL FMMUL(QY,C,NPD,NPD,NND,Z1)
      CALL FTMUL(C,Z1,NPD,NND,NND,Z2)
      CALL TFRMTX(Z2,X,NND,NND,2)
      L1=LADDR(NNPR,NND+1,NND+1)
      CALL TFRM7X(RY,X(L1),NRD,NRD,2)
      L2=LADDR(NNPR,NND+1,1)
      IF(NODY.EQ.0) GO TO 5
      CALL ZPART(X(L2),NRD,NND,NNPR)
      GO TO 15
    5 CALL FTMUL(D,Z1,NPD,NRD,NND,Z2)
      CALL TFRMTX(Z2,X(L2),NRD,NND,2)
      CALL FMMUL(QY,D,NPD,NPD,NRD,Z1)
      CALL FTMUL(D,Z1,NPD,NRD,NRD,Z2)
      L2=0
      DO 12 I=1,NRD
      L1=LADDR(NNPR,NND+1,NND+I)
      DO 12 J=1,NRD
      L1=L1+1
      L2=L2+1
      X(L1)=X(L1)+Z2(L2)
   12 L1=L1+1
   15 DO 20 I=1,NND
      L1=LADDR(NNPR,NND+1,I)
      L2=LADDR(NNPR,I,NND+1)
      DO 20 J=1,NRD
      X(L2)=X(L1)
      L1=L1+1
```

```
 20   L2=L2+NNPR
      RETURN
C END SUBROUTINE FORMX
      END
      SUBROUTINE PXUP(PHIDL,BDEL,X,U,S,BUIBT,UIST,PHIP,XP,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      DIMENSION PHIDL(1),BDEL(1),X(1),U(1),S(1),BUIBT(1),UIST(1),
     1 PHIP(1),XP(1),ZM1(1)
      CALL EQUATE(PHIP,U,NRD,NRD)
      CALL GMINV(NRD,NRD,PHIP,ZM1,MR,1)
      CALL MAT3(NDIM,NRD,BDEL,ZM1,BUIBT)
      CALL MAT5(ZM1,S,NRD,NRD,NDIM,UIST)
      CALL MMUL(BDEL,UIST,NDIM,NRD,NDIM,ZM1)
      CALL MADD1(NDIM,NDIM,PHIDL,ZM1,PHIP,-1.)
      CALL MMUL(S,UIST,NDIM,NRD,NDIM,ZM1)
      CALL MADD1(NDIM,NDIM,X,ZM1,XP,-1.)
      RETURN
C END SUBROUTINE PXUP
      END
      SUBROUTINE GCSTAR(PHIP,BDEL,U,RK,UIST,GCS,ZM1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      DIMENSION PHIP(1),BDEL(1),U(1),RK(1),UIST(1),GCS(1),ZM1(1)
      CALL MAT3A(NRD,NDIM,BDEL,RK,ZM1)
      CALL MADD1(NRD,NRD,ZM1,U,ZM1,1.)
      CALL GMINV(NRD,NRD,ZM1,U,MR,1)
      CALL MAT5(U,BDEL,NRD,NRD,NDIM,ZM1)
      CALL MAT1(ZM1,RK,NRD,NDIM,NDIM,GCS)
      CALL MMUL(GCS,PHIP,NRD,NDIM,NDIM,ZM1)
      CALL MADD1(NRD,NDIM,ZM1,UIST,GCS,1.)
      WRITE(KLIST,101)
 101  FORMAT("0REG/PI GAIN MATRIX--GCS"/)
      CALL MATIO(GCS,NRD,NDIM,3)
      RETURN
C END SUBROUTINE GCSTAR
      END
      SUBROUTINE SCGT
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,LKXA12,LKXA13
      COMMON/CCGT/NVCGT,CGT(1)
      IF(NEWCM) 20,20,15
 15   NSIZE=(NND+2*NPD)*(NNC+NRC+NDD)
      IF(NSIZE.LE.NVCGT) GO TO 16
      WRITE 106,NSIZE
      LABORT=NSIZE
      RETURN
   16 IF(NND.GE.NNC) GO TO 17
      WRITE 107
      GO TO 18
   17 IF(NND.GE.NDD) GO TO 19
      WRITE 108
   18 LABORT=-1
      RETURN
   19 NEWCM=0
      LA11=1
      LA13=LA11+NND*NNC
      LA21=LA13+NND*NDD
      LA23=LA21+NPD*NNC
      LA12=LA23+NPD*NDD
      LA22=LA12+NND*NRC
      LKXA11=LA22+NPD*NRC
      LKXA12=LKXA11+NPD*NNC
      LKXA13=LKXA12+NPD*NRC
      CALL CGTA(CGT(LA11),CGT(LA13),CGT(LA21),CGT(LA23),CGT(LA12),
     1 CGT(LA22),ZM1,ZM2)
   20 CALL CGTKX(CGT(LA11),CGT(LA13),CGT(LA21),CGT(LA23),CGT(LA12),
     1 CGT(LA22),CGT(LKXA11),CGT(LKXA12),CGT(LKXA13),RPI(LKX))
      LFLCGT=1
  106 FORMAT("0INSUFFICIENT MEMORY /CCGT/, NEED: ",I4)
  107 FORMAT("0FEWER DESIGN MODEL THAN COMMAND MODEL STATES")
  108 FORMAT("FEWER DESIGN MODEL THAN DISTURBANCE MODEL STATES")
      RETURN
C END SUBROUTINE SCGT
      END
      SUBROUTINE CGTA(A11,A13,A21,A23,A12,A22,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      DIMENSION A11(1),A13(1),A21(1),A23(1),A12(1),A22(1),ZM1(1),ZM2(1)
      NDIM=NND
      NDIM1=NDIM+1
      CALL TFRMTX(CM,ZM1,NNC,NNC,2)
      CALL SUPI(ZM1,NNC,NDIM)
      CALL FMMUL(CTL(LPI12),CM(LCC),NND,NRD,NNC,ZM2)
      CALL MSCALE(ZM2,ZM2,NND,NNC,-1.)
      NB=MAX0(NDD,NNC)
      L2=1+NND*NND
      L3=L2+NND*NB
      L4=L3+NND*NB
      L5=L4+NND*NND
      L6=L5+NND*NND
      NSIZE=L6+NPD*NNC-1
      IF(NSIZE.LE.NVSM) GO TO 1
      WRITE 102,NSIZE
  102 FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
      LABORT=NSIZE
      RETURN
    1 CALL AXBMXC(CTL(LPI11),NND,ZM1,NNC,ZM2,A11,SM,
     1 SM(L2),SM(L3),SM(L4),SM(L5))
      CALL MMUL(A11,ZM1,NND,NNC,NNC,ZM2)
      CALL FMMUL(CTL(LPI21),ZM2,NPD,NND,NNC,A21)
      CALL FMMUL(CTL(LPI22),CM(LCC),NPD,NRD,NNC,SM(L6))
      CALL FMMUL(A11,CM(LBDC),NND,NNC,NRC,SM)
      CALL FMMUL(CTL(LPI11),SM,NND,NND,NRC,A12)
      CALL FMMUL(CTL(LPI21),SM,NPD,NND,NRC,A22)
      IF(NODC.EQ.1) GO TO 2
      CALL FMMUL(CTL(LPI12),CM(LDC),NND,NRD,NRC,SM(L2))
      CALL FMADD(A12,SM(L2),NND,NRC,A12)
      CALL FMMUL(CTL(LPI22),CM(LDC),NPD,NRD,NRC,SM(L2))
      CALL FMADD(A22,SM(L2),NPD,NRC,A22)
    2 IF(NDD.EQ.0) GO TO 15
      CALL MMUL(CTL(LPI11),DM(LEX),NND,NND,NDD,ZM2)
      IF(NOEY.EQ.1) GO TO 5
      CALL FMMUL(CTL(LPI12),DM(LEY),NND,NRD,NDD,ZM1)
      CALL MADD1(NND,NDD,ZM1,ZM2,ZM2,1.)
    5 CALL TFRMTX(DM(LPHD),ZM1,NDD,NDD,2)
      CALL SUBI(ZM1,NDD,NDIM)
      CALL AXBMXC(CTL(LPI11),NND,ZM1,NDD,ZM2,A13,SM,
     1 SM(L2),SM(L3),SM(L4),SM(L5))
      CALL MMUL(A13,ZM1,NND,NDD,NDD,ZM2)
      CALL MADD1(NND,NDD,ZM2,DM(LEX),ZM2,-1.)
      CALL FMMUL(CTL(LPI21),ZM2,NPD,NND,NDD,A23)
   15 NDIM=NPD
      NDIM1=NDIM+1
      CALL MADD1(NPD,NNC,A21,SM(L6),A21,1.)
      IF(NOEY.EQ.1) GO TO 20
      CALL MMUL(CTL(LPI22),DM(LEY),NPD,NRD,NDD,ZM1)
      CALL MADD1(NPD,NDD,A23,ZM1,A23,-1.)
   20 CALL MATLST(A11,NND,NNC,"A11",KLIST)
      CALL MATLST(A21,NPD,NNC,"A21",KLIST)
      CALL MATLST(A12,NND,NRC,"A12",KLIST)
      CALL MATLST(A22,NPD,NRC,"A22",KLIST)
      IF(NDD.GT.0) GO TO 25
      WRITE(KLIST,101)
  101 FORMAT("0MATRICES A13 AND A23 ARE ZERO")
      RETURN
   25 CALL MATLST(A13,NND,NDD,"A13",KLIST)
      CALL MATLST(A23,NPD,NDD,"A23",KLIST)
      RETURN
C END SUBROUTINE CGTA
      END
      SUBROUTINE AXBMXC(A,NA,B,NB,C,X,AU,BU,R,Z1,Z2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
```

AD-A138 309 ROBUST MULTIVARIABLE CONTROLLER DESIGN VIA IMPLICIT 4/4
MODEL-FOLLOWING METHODS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W G MILLER
UNCLASSIFIED DEC 83 AFIT/GE/EE/83D-48 F/G 14/2 NL

END
FILMED
4-84
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
      DIMENSION A(1),B(1),C(1),X(1),AU(1),BU(1),R(1),Z1(1),Z2(1)
      DATA EMAX,ITMAX/1.E-6,3/
      CALL TRANS1(NA,A,Z1)
      CALL EIGEN(NA,Z1,Z2,Z2(NDIM1),AU,1)
      CALL TRANS1(NA,COM1,Z1)
      CALL EIGEN(NB,B,Z2,Z2(NDIM1),BU,1)
      CALL EQUATE(R,C,NA,NB)
      IT=0
 10   CALL MAT4A(AU,R,NA,NA,NB,Z2)
      CALL MAT1(Z2,BU,NA,NB,NB,R)
      CALL SLVSHR(Z1,NA,COM1,NB,R,NDIM)
      CALL MAT4(R,BU,NA,NB,NB,Z2)
      CALL MAT1(AU,Z2,NA,NA,NB,R)
      IF(IT.GT.0) GO TO 15
      CALL EQUATE(X,R,NA,NB)
      GO TO 30
 15   CALL MADD1(NA,NB,X,R,X,1.)
      CALL ENORM(R,NA,NB,EN)
      IF(EN.LE.EMAX) RETURN
      IF(IT.LT.ITMAX) GO TO 30
      WRITE(KLIST,101) EN
      WRITE(KTERM,101) EN
 101  FORMAT("0SOLUTION ERROR FOR 'A'(CGT) AFTER 3 ITERATIONS = ",1PE15.
     17)
      RETURN
 30   CALL MAT1(A,X,NA,NA,NB,Z2)
      CALL MAT1(Z2,B,NA,NB,NB,R)
      CALL MADD1(NA,NB,X,R,R,-1.)
      CALL MADD1(NA,NB,R,C,R,1.)
      IT=IT+1
      GO TO 10
C END SUBROUTINE AXBMXC
      END
      SUBROUTINE SLVSHR(A,NA,B,NB,C,ND)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      DIMENSION A(ND,1),B(ND,1),C(ND,1),V(16),W(4)
      L=1
 5    LM1=L-1
      DL=1
      IF(L.EQ.NB) GO TO 8
      IF(B(L+1,L).NE.0.) DL=2
 8    LL=LM1+DL
      K=1
 10   KM1=K-1
      DK=1
      IF(K.EQ.NA) GO TO 12
      IF(A(K,K+1).NE.0.) DK=2
 12   KK=KM1+DK
      AKK=A(K,K)
      BLL=B(L,L)
      IF(DL.EQ.2) GO TO 35
      IF(DK.EQ.2) GO TO 20
      IF(L.EQ.1) GO TO 13
```

```
12500=      C(K,L)=C(K,L)-AKK*DOT3(LM1,C(K,1),B(1,L))
12510= 13   IF(K.EQ.1) GO TO 18
12520=      DO 15 I=1,KM1
12530= 15   C(K,L)=C(K,L)-A(K,I)*DOT3(L,C(I,1),B(1,L))
12540= 18   V(1)=AKK*BLL-1.
12550=      IF(V(1).EQ.0.) GO TO 99
12560=      C(K,L)=C(K,L)/V(1)
12570=      GO TO 95
12580= 20   IF(L.EQ.1) GO TO 22
12590=      I1=K
12600=      I2=KK
12610=      I3=LM1
12620=      GO TO 24
12630= 22   IF(K.EQ.1) GO TO 30
12640=      I1=1
12650=      I2=KM1
12660=      I3=L
12670= 24   DO 28 I=I1,I2
12680=      V(1)=DOT3(I3,C(I,1),B(1,L))
12690=      C(K,L)=C(K,L)-A(K,I)*V(1)
12700= 28   C(KK,L)=C(KK,L)-A(KK,I)*V(1)
12710=      IF(I1.EQ.K) GO TO 22
12720= 30   V(1)=AKK*BLL-1.
12730=      V(2)=A(KK,K)*BLL
12740=      V(3)=A(K,KK)*BLL
12750=      V(4)=A(KK,KK)*BLL-1.
12760=      V(5)=1./(V(1)*V(4)-V(2)*V(3))
12770=      V(6)=V(5)*(C(K,L)*V(4)-V(3)*C(KK,L))
12780=      C(KK,L)=V(5)*(V(1)*C(KK,L)-V(2)*C(K,L))
12790=      C(K,L)=V(6)
12800=      GO TO 95
12810= 35   IF(DK.EQ.2) GO TO 50
12820=      IF(L.EQ.1) GO TO 38
12830=      I1=K
12840=      I2=K
12850=      I3=LM1
12860=      GO TO 40
12870= 38   IF(K.EQ.1) GO TO 45
12880=      I1=1
12890=      I2=KM1
12900=      I3=LL
12910= 40   DO 42 I=I1,I2
12920=      C(K,L)=C(K,L)-A(K,I)*DOT3(I3,C(I,1),B(1,L))
12930= 42   C(K,LL)=C(K,LL)-A(K,I)*DOT3(I3,C(I,1),B(1,LL))
12940=      IF(I1.EQ.K) GO TO 38
12950= 45   V(1)=AKK*BLL-1.
12960=      V(2)=AKK*B(L,LL)
12970=      V(3)=AKK*B(LL,L)
12980=      V(4)=AKK*B(LL,LL)-1.
12990= 48   V(5)=1./(V(1)*V(4)-V(2)*V(3))
13000=      V(6)=V(5)*(C(K,L)*V(4)-V(3)*C(K,LL))
13010=      C(K,LL)=V(5)*(V(1)*C(K,LL)-V(2)*C(K,L))
13020=      C(K,L)=V(6)
13030=      GO TO 95
```

```
 50   IF(L.EQ.1) GO TO 55
      V(1)=DOT3(LM1,C(K,1),B(1,L))
      V(2)=DOT3(LM1,C(KK,1),B(1,L))
      V(3)=DOT3(LM1,C(K,1),B(1,LL))
      V(4)=DOT3(LM1,C(KK,1),B(1,LL))
      C(K,L)=C(K,L)-AKK*V(1)-A(K,KK)*V(2)
      C(KK,L)=C(KK,L)-A(KK,K)*V(1)-A(KK,KK)*V(2)
      C(K,LL)=C(K,LL)-AKK*V(3)-A(K,KK)*V(4)
      C(KK,LL)=C(KK,LL)-A(KK,K)*V(3)-A(KK,KK)*V(4)
 55   IF(K.EQ.1) GO TO 65
      DO 60 I=1,KM1
      V(1)=DOT3(LL,C(I,1),B(1,L))
      V(2)=DOT3(LL,C(I,1),B(1,LL))
      C(K,L)=C(K,L)-A(K,I)*V(1)
      C(KK,L)=C(KK,L)-A(KK,I)*V(1)
      C(K,LL)=C(K,LL)-A(K,I)*V(2)
 60   C(KK,LL)=C(KK,LL)-A(KK,I)*V(2)
 65   V(1)=AKK*BLL-1.
      V(2)=A(KK,K)*BLL
      V(3)=AKK*B(L,LL)
      V(4)=A(KK,K)*B(L,LL)
      V(5)=A(K,KK)*BLL
      V(6)=A(KK,KK)*BLL-1.
      V(7)=A(K,KK)*B(L,LL)
      V(8)=A(KK,KK)*B(L,LL)
      V(9)=AKK*B(LL,L)
      V(10)=A(KK,K)*B(LL,L)
      V(11)=AKK*B(LL,LL)-1.
      V(12)=A(KK,K)*B(LL,LL)
      V(13)=A(K,KK)*B(LL,L)
      V(14)=A(KK,KK)*B(LL,L)
      V(15)=A(K,KK)*B(LL,LL)
      V(16)=A(KK,KK)*B(LL,LL)-1.
      W(1)=C(K,L)
      W(2)=C(KK,L)
      W(3)=C(K,LL)
      W(4)=C(KK,LL)
      NDS=NDIM
      NDIM=4
      NDIM1=NDIM+1
      CALL DOOLIT(4,V,W,1,ISG)
      NDIM=NDS
      NDIM1=NDIM+1
 95   K=K+DK
      IF(K.LE.NA) GO TO 10
      L=L+DL
      IF(L.LE.NB) GO TO 5
      RETURN
 99   WRITE(KOUT,101)
      RETURN
 101  FORMAT("0* * * ERROR IN CGT SOLUTION: A11->A23")
C END SUBROUTINE SLVSHR
      END
      SUBROUTINE ENORM(A,NR,NC,ENRM)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      DIMENSION A(1)
      ENRM=0.
      NE=NC*NDIM
      DO 10 I=1,NR
      DO 10 J=I,NE,NDIM
   10 ENRM=ENRM+A(J)*A(J)
      ENRM=SQRT(ENRM)
      RETURN
C END SUBROUTINE ENORM
      END
      SUBROUTINE CGTKX(A11,A13,A21,A23,A12,A22,RKXA11,RKXA12,RKXA13,RKX)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      DIMENSION A11(1),A13(1),A21(1),A23(1),A12(1),A22(1),
     1 RKXA11(1),RKXA12(1),RKXA13(1),RKX(1)
      NDIM=NRD
      NDIM1=NDIM+1
      CALL FMMUL(RKX,A11,NRD,NND,NNC,RKXA11)
      CALL MADD1(NRD,NNC,RKXA11,A21,RKXA11,1.)
      CALL MATLST(RKXA11,NRD,NNC,"KXN",KLIST)
      CALL MATLST(RKXA11,NRD,NNC,"KXN",KTERM)
      CALL FMMUL(RKX,A12,NRD,NND,NRC,RKXA12)
      CALL MADD1(NRD,NRC,RKXA12,A22,RKXA12,1.)
      CALL MATLST(RKXA12,NRD,NRC,"KXU",KLIST)
      CALL MATLST(RKXA12,NRD,NRC,"KXU",KTERM)
      IF(NDD.LT.1) RETURN
      CALL FMMUL(RKX,A13,NRD,NND,NDD,RKXA13)
      CALL MADD1(NRD,NDD,RKXA13,A23,RKXA13,1.)
      CALL MATLST(RKXA13,NRD,NDD,"KXN",KLIST)
      CALL MATLST(RKXA13,NRD,NDD,"KXN",KTERM)
      RETURN
C END SUBROUTINE CGTKX
      END
      SUBROUTINE CEVAL
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LREGPI/LXDN,LUDN,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION NPLOT(2),NVPLOT(10),NS(6),LSCL(2),ITITLE(5)
      DATA NO/1HN/
      WRITE(KLIST,110)
  110 FORMAT(////11X,5("* "),"CONTROLLER EVALUATION",5(" *")////)
      IPOLE=1
   10 NVOUT=NRD+NPD+1
```

```
      IF(LFLCGT) 17,17,15
   15 WRITE 106
      READ*,IUM,VUM
      IF(IUM.LT.1) GO TO 70
      IF(IUM.GT.NRC) GO TO 15
      NVOUT=NVOUT+NPC
      NP=NNC
      GO TO 18
   17 IF(IPOLE.EQ.1) CALL POLES(RPI(LPHCL),NNPR,4,ZM1,ZM2)
      NP=0
   18 CALL VOUTIC(SM,NVPLOT,NPLOT,NVOUT,LSCL)
      IF(NVOUT.EQ.0) GO TO 70
   20 WRITE 100
      READ*,TEND
      IF(TEND) 20,20,25
   25 LVX0=NVOUT+1
      LX0=LVX0+NVOUT
      LX1=LX0+NPLD
      LXM0=LX1+NPLD
      LXM1=LXM0+NP
      NP=LXM1+NP
      DO 26 I=LVX0,NP
   26 SM(I)=0.
      CALL CTRESP(SM(LVX0),SM,SM(LX0),SM(LX1),SM(LXM0),SM(LXM1),
     1 ZM1,NVOUT,TEND,IUM,VUM,NST)
      WRITE(KTERM,101)
      READ(KIN,102) ITITLE
      M=50*NST
      DO 40 I=1,2
      NS(1)=1
      DO 28 J=2,6
   28 NS(J)=NS(J-1)+51
      NP=NPLOT(I)
      IF(NP.EQ.0) GO TO 40
      NPP1=NP+1
      REWIND KPLOT
      NSV=5*I-4
      CALL RPLOTF(ZM1,NVOUT,IERR)
      CALL STRPLT(SM,ZM1,NS,NVPLOT(NSV),NP,NVOUT)
      DO 35 J=1,M
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 40
      IF(MOD(J,NST).NE.0) GO TO 35
      DO 30 K=1,NPP1
   30 NS(K)=NS(K)+1
      CALL STRPLT(SM,ZM1,NS,NVPLOT(NSV),NP,NVOUT)
   35 CONTINUE
      CALL PLOTLP(51,NP,SM,LSCL(I),1,0,KTERM,ITITLE)
   40 CONTINUE
      NVM=NVOUT-1
      M=NVM/5
      NE=5*M
      IF(M.EQ.0) GO TO 56
      DO 55 I=1,NE,5
      NS(1)=1
      DO 42 J=2,6
42    NS(J)=NS(J-1)+101
      REWIND KPLOT
      NVS=I-1
      DO 45 J=1,5
45    NVPLOT(J)=NVS+J
      DO 50 J=1,101
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 55
      CALL STRPLT(SM,ZM1,NS,NVPLOT,5,NVOUT)
      DO 48 K=1,6
48    NS(K)=NS(K)+1
50    CONTINUE
      CALL PLOTLP(101,5,SM,1,1,1,KLIST,ITITLE)
55    CONTINUE
56    NVM=NVM-NE
      IF(NVM.LT.1) GO TO 70
      NPP1=NVM+1
      NS(1)=1
      DO 57 I=2,6
57    NS(I)=NS(I-1)+101
      DO 58 I=1,NVM
58    NVPLOT(I)=NE+I
      REWIND KPLOT
      DO 65 I=1,101
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 70
      CALL STRPLT(SM,ZM1,NS,NVPLOT,NVM,NVOUT)
      DO 60 J=1,NPP1
60    NS(J)=NS(J)+1
65    CONTINUE
      CALL PLOTLP(101,NVM,SM,1,1,1,KLIST,ITITLE)
70    WRITE 104
      READ 111,IANS
      IF(IANS.EQ.NO) RETURN
      IPOLE=0
      GO TO 10
101   FORMAT(" +",10("-")," ENTER TITLE IN GIVEN FIELD ",10("-"),"+"/)
102   FORMAT(5A10)
104   FORMAT("0MORE TIME RESPONSE RUNS (Y OR N) >")
106   FORMAT("0ENTER MODEL INPUT AND STEP VALUE : 1 >")
108   FORMAT(" ENTER TIME DURATION FOR RESPONSE, IN SECONDS >")
111   FORMAT(A3)
C END SUBROUTINE CEVAL
      END
      SUBROUTINE VOUTIC(VIC,NVPLOT,NPLOT,NVOUT,LSCL)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      DIMENSION NPLOT(1),NVPLOT(1),VIC(1),IOUT(5),LSCL(2)
      DATA IOUT/1HX,1HY,1HU,1HM,1HD/
      IF(LTEVAL) 2,2,5
 2    NVS=NND
      NV=NPLD
      GO TO 8
 5    NV=NNT
      NVS=NV
 8    NVOUT=NVOUT+NV
      DO 9 I=1,NVOUT
 9    VIC(I)=0.
      NVU=NV+NPD
      NVM=NVU+NRD
 10   WRITE 101,NVS
 101  FORMAT("0ENTER STATE AND IC VALUE (0/ TERMINATES): ",I2," >")
 12   READ*,IV,V
      IF(IV.LT.1) GO TO 15
      IF(IV.GT.NVS) GO TO 10
      VIC(IV)=V
      GO TO 12
 15   IF((LFLCGT.LT.1).OR.(LTEVAL.EQ.1).OR.(NDD.LT.1)) GO TO 25
      LD=1
 18   WRITE 102,NDD
 102  FORMAT(" ENTER DISTURBANCE IC VALUE (0/ TERMINATES): ",I2," >")
 20   READ*,IV,V
      IF(IV.LT.1) GO TO 26
      IF(IV.GT.NDD) GO TO 18
      VIC(NND+IV)=V
      GO TO 20
 25   LD=0
 26   WRITE 103
 103  FORMAT(" 2 PLOTS OF 5 VARIABLES MAY BE PRINTED AT THE TERMINAL --
     1SPECIFY NUMBER FOR EACH (N1,N2) >")
      READ*,NPLOT(1),NPLOT(2)
      IF(NPLOT(1).GT.5) NPLOT(1)=5
      IF(NPLOT(2).GT.5) NPLOT(2)=5
      IF((NPLOT(1).GT.0).OR.(NPLOT(2).GT.0)) GO TO 27
      NVOUT=0
      RETURN
 27   WRITE 104
 104  FORMAT(" ENTER OUTPUTS BY TYPE AND INDEX IN 2 ENTRIES--TYPES ARE"/
     1 " STATE : 'X'"/" OUTPUT : 'Y'"/" INPUT : 'U'")
      IF(LFLCGT) 30,30,28
 28   WRITE 105
 105  FORMAT(" MODEL : 'M'")
      IF(LD.EQ.1) WRITE 106
 106  FORMAT(" DISTURBANCE : 'D'")
 30   DO 40 I=1,2
      NC=NPLOT(I)
      IF(NC.LT.1) GO TO 40
      LSCL(I)=1
      NS=5*(I-1)
      WRITE 107,I
 107  FORMAT("0PLOT ",I2)
      DO 39 J=1,NC
      NSP=NS+J
```

```
31   WRITE 100,J
100  FORMAT(" OUTPUT ",I2," >")
     READ 111,IV
     WRITE 113
113  FORMAT(11X,">")
     READ*,IO
     IF(IV.NE.IOUT(1)) GO TO 32
     IF(IO.GT.NVS) GO TO 38
     NVPLOT(NSP)=IO
     GO TO 39
32   IF(IV.NE.IOUT(2)) GO TO 321
     IF(IO.GT.NPD) GO TO 38
     NVPLOT(NSP)=NV+IO
     GO TO 39
321  IF(IV.NE.IOUT(3)) GO TO 33
     IF(IO.GT.NRD) GO TO 38
     NVPLOT(NSP)=NVU+IO
     GO TO 39
33   IF(LFLCGT.LT.1) GO TO 31
     IF(IV.NE.IOUT(4)) GO TO 34
     IF(IO.GT.NPC) GO TO 38
     NVPLOT(NSP)=NVM+IO
     LSCL(I)=-1
     GO TO 39
34   IF(LD.NE.1) GO TO 31
     IF(IV.NE.IOUT(5)) GO TO 31
     IF(IO.GT.NDD) GO TO 38
     NVPLOT(NSP)=NVS+IO
     GO TO 39
38   WRITE 109
109  FORMAT(" INDEX TOO LARGE")
     GO TO 31
39   CONTINUE
40   CONTINUE
     NVM1=NVOUT-1
     NP=0
     DO 50 I=1,NVM1
     M=MOD((I-1),5)+1
     IF(M.GT.1) GO TO 41
     NP=NP+1
     WRITE(KLIST,110) NP
110  FORMAT("0PLOT ",I2)
41   IF(I.GT.NVS) GO TO 42
     IV=IOUT(1)
     IO=I
     GO TO 45
42   IF(I.GT.NV) GO TO 43
     IV=IOUT(5)
     IO=I-NVS
     GO TO 45
43   IF(I.GT.NVU) GO TO 441
     IV=IOUT(2)
     IO=I-NV
     GO TO 45
```

```
  441 IF(I.GT.NVM) GO TO 44
      IV=IOUT(3)
      IO=I-NVU
      GO TO 45
   44 IV=IOUT(4)
      IO=I-NVM
   45 WRITE(KLIST,112) M,IV,IO
   50 CONTINUE
      RETURN
  111 FORMAT(A1)
  112 FORMAT("    OUTPUT ",I2,": ",A1,I2)
C END SUBROUTINE VOUTIC
      END
      SUBROUTINE CTRESP(VX0,VX1,X0,X1,XM0,XM1,ZM1,NVOUT,TEND,IUM,VUM,
     1 NST)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIND/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LGN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDINC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CNDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LREGPI/LXDM,LUDM,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      DIMENSION VX0(1),VX1(1),X0(1),X1(1),XM0(1),XM1(1),ZM1(1)
      NSTPO=.01*TEND/TSAMP+.5
      NST=2
      IF(NSTPO.GE.1) GO TO 1
      NSTPO=1
      NST=1
    1 NSTEPS=100*NSTPO
      IF(LFLCGT.EQ.0) GO TO 2
      LMO=NVOUT-NPC
      IF(NDD.EQ.0) GO TO 4
      LDCGT=1
      GO TO 5
    2 LMO=NVOUT
    4 LDCGT=0
    5 LU=LMO-NRD
      LSO=LU-NPD
      NVX=LSO-1
      IF(LTEVAL)6,6,10
    6 DO 7 I=1,NVX
    7 X1(I)=VX1(I)
      GO TO 12
   10 CALL XFDT(VX1,X1,LDCGT)
   12 NNDP1=NND+1
      REWIND KPLOT
      CALL YDSN(X1,VX1(LU),DM(LC),DM(LDY),LDCGT,VX1(LSO))
      IF(LFLCGT.EQ.1) CALL YCMD(XM1,IUM,VUM,CM(LCC),CM(LDC),
     1 VX1(LMO))
```

```
      CALL WPLOTF(VX1,NVOUT)
      DO 100 IT=1,NSTEPS
      CALL URPI(RPI(LKX),RPI(LKZ),DM(LC),DM(LDY),X0,X1,VX0(LU),VX1(LU))
      IF(LFLCGT) 20,20,15
   15 CALL UCGT(VX0(LU),VX1(LU),XM0,XM1,X0(NNDP1),ZM1,IUM,VUM,IT)
      CALL CUPDAT(XM0,XM1,IUM,VUM)
   20 CALL FTMTX(VX1(LU),VX0(LU),NRD,1)
      CALL FTMTX(X1,X0,NPLD,1)
      IF(LTEVAL) 25,25,30
   25 CALL DUPDAT(DM(LPHI),DM(LBD),DM(LPHD),DM(LEX),X0,X1,
     1 VX1,VX0(LU),LDCGT,NNDP1)
      GO TO 35
   30 CALL TUPDAT(TM(LPHT),TM(LBDT),VX0,VX1,VX0(LU))
      CALL XFDT(VX1,X1,LDCGT)
   35 IF(MOD(IT,NSTPO).NE.0) GO TO 100
      VX1(NVOUT)=TSAMP*FLOAT(IT)
      CALL YDSN(X1,VX1(LU),DM(LC),DM(LDY),LDCGT,VX1(LSO))
      IF(LFLCGT.EQ.1) CALL YCMD(XM1,IUM,VUM,CM(LCC),CM(LDC),
     1 VX1(LMO))
      CALL WPLOTF(VX1,NVOUT)
  100 CONTINUE
      ENDFILE KPLOT
      RETURN
C END SUBROUTINE CTRESP
      END
      SUBROUTINE DUPDAT(PHI,BD,PHID,EX,X0,X1,VX1,U0,LDCGT,NNDP1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NND,NNDD,NPLD,NWPNND,NNPR
      DIMENSION PHI(1),BD(1),PHID(1),EX(1),X0(1),X1(1),VX1(1),U0(1)
      NDIM=NND
      NDIM1=NDIM+1
      CALL FMMUL(BD,U0,NND,NRD,1,X1)
      CALL MMULS(PHI,X0,NND,NND,1,X1)
      IF(LDCGT.EQ.0) GO TO 10
      CALL FMMUL(PHID,X0(NNDP1),NDD,NDD,1,X1(NNDP1))
      CALL MMULS(EX,X1(NNDP1),NND,NDD,1,X1)
   10 CALL FTMTX(X1,VX1,NPLD,1)
      RETURN
C END SUBROUTINE DUPDAT
      END
      SUBROUTINE CUPDAT(XM0,XM1,IUM,VUM)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      DIMENSION XM0(1),XM1(1)
      NDIM=NNC
      NDIM1=NDIM+1
      CALL FTMTX(XM1,XM0,NNC,1)
      L1=LADDR(NNC,1,IUM)+LBDC-1
      CALL VSCALE(XM1,CM(L1),NNC,VUM)
      CALL MMULS(CM(LPHC),XM0,NNC,NNC,1,XM1)
      RETURN
C END SUBROUTINE CUPDAT
```

```
      END
      SUBROUTINE TUPDAT(PHI,BD,VX0,VX1,U0)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      DIMENSION PHI(1),BD(1),VX0(1),VX1(1),U0(1)
      NDIM=NNT
      NDIM1=NDIM+1
      CALL FTMTX(VX1,VX0,NNT,1)
      CALL FMMUL(BD,U0,NNT,NRT,1,VX1)
      CALL MMULS(PHI,VX0,NNT,NNT,1,VX1)
      RETURN
C END SUBROUTINE TUPDAT
      END
      SUBROUTINE XFDT(V,X,LDCGT)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      DIMENSION V(1),X(1)
      CALL FMMUL(TM(LTDT),V,NND,NNT,1,X)
      IF(LDCGT.EQ.0) RETURN
      CALL FMMUL(TM(LTNT),V,NDD,NNT,1,X(NND+1))
      RETURN
C END SUBROUTINE XFDT
      END
      SUBROUTINE URPI(RKX,RKZ,C,DY,X0,X1,U0,U1)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      DIMENSION RKX(1),RKZ(1),C(1),DY(1),X0(1),X1(1),U0(1),U1(1)
      CALL YDSN(X0,U0,C,DY,0,U1)
 10   CALL VSCALE(U1,U1,NRD,-1.)
      CALL MMULS(RKZ,U1,NRD,NRD,1,U0)
      DO 12 I=1,NPLD
 12   X0(I)=X0(I)-X1(I)
      CALL FMMUL(RKX,X0,NRD,NND,1,U1)
      CALL VADD(NRD,1.,U1,U0)
      RETURN
C END SUBROUTINE URPI
      END
      SUBROUTINE UCGT(U0,U1,XM0,XM1,DDIF,ZM1,IUM,VUM,IT)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,LKXA12,LKXA13
      COMMON/CCGT/NVCGT,CGT(1)
      DIMENSION U0(1),U1(1),XM0(1),XM1(1),DDIF(1),ZM1(1)
      CALL YCMD(XM0,IUM,VUM,CM(LCC),CM(LDC),U0)
      IF(IT.GT.1) GO TO 10
      I=LKXA12+LADDR(NPD,1,IUM)-1
      CALL MADD1(NPD,1,U1,CGT(I),U1,VUM)
 10   CALL MMULS(RPI(LKZ),U0,NDIM,NDIM,1,U1)
      DO 12 I=1,NNC
 12   XM0(I)=XM1(I)-XM0(I)
      CALL FMMUL(CGT(LKXA11),XM0,NPD,NNC,1,U0)
      CALL VADD(NDIM,1.,U1,U0)
      IF(NDD.EQ.0) RETURN
      DO 14 I=1,NDD
 14   DDIF(I)=-DDIF(I)
      CALL MMULS(CGT(LKXA13),DDIF,NPD,NDD,1,U1)
      RETURN
C END SUBROUTINE UCGT
      END
      SUBROUTINE YDSN(X,U,C,D,LDCGT,Y)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      DIMENSION X(1),U(1),C(1),D(1),Y(1)
      NDIM=NPD
      NDIM1=NDIM+1
      CALL FMMUL(C,X,NPD,NND,1,Y)
      IF(NODY.EQ.1) GO TO 10
      CALL MMULS(D,U,NPD,NRD,1,Y)
 10   IF((LDCGT.EQ.0).OR.(NOEY.EQ.1)) RETURN
      CALL MMULS(DM(LEY),X(NND+1),NPD,NDD,1,Y)
      RETURN
C END SUBROUTINE YDSN
      END
      SUBROUTINE YCMD(X,IU,VU,C,D,Y)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      DIMENSION X(1),C(1),D(1),Y(1)
      NDIM=NPC
      NDIM1=NDIM+1
      CALL FMMUL(C,X,NPC,NNC,1,Y)
      IF(NODC.EQ.1) RETURN
      L1=LADDR(NPC,1,IU)
      CALL MADD1(NPC,1,Y,D(L1),Y,VU)
      RETURN
C END SUBROUTINE YCMD
      END
      SUBROUTINE FLTRK(IFLTR)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/MAIN2/COM2(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
```

```
18440=      COMMON/CKF/NVFLT,FLT(1)
18450=      IF(NWPNMD.GT.0) GO TO 1
18460=      WRITE(KTERM,100)
18470= 100  FORMAT("0NO DRIVING NOISES - - FILTER DESIGN ABORT")
18480=      RETURN
18490= 1    IF(NMD.GT.0) GO TO 2
18500=      WRITE(KTERM,109)
18510= 109  FORMAT("0NO MEASUREMENTS - - FILTER DESIGN ABORT")
18520=      RETURN
18530= 2    WRITE(KLIST,110)
18540= 110  FORMAT(////11X,5("* "),"FILTER DESIGN",5(" *")////)
18550=      NSIZE=NPLD*(1+NPLD+NMD)
18560=      IF(NSIZE.LE.NVFLT) GO TO 3
18570=      WRITE 101,NSIZE
18580= 101  FORMAT("0INSUFFICIENT MEMORY /CKF/, NEED: ",I4)
18590=      LABORT=NSIZE
18600=      RETURN
18610= 3    NDIM=NPLD
18620=      NDIM1=NDIM+1
18630= 5    IF(NWD.EQ.0) GO TO 12
18640=      IF(IFLTR.LE.0) GO TO 6
18650=      WRITE 105,NWD
18660= 105  FORMAT(" ENTER STATE NOISE STRENGTHS: ",I2)
18670=      CALL RDWGTS(DM(LQ),NWD,0)
18680= 6    CALL DVCTOR(NWD,DM(LQ),ZM1)
18690=      CALL MATLST(ZM1,NWD,1,"Q",KTERM)
18700= 10   CALL MATLST(DM(LQ),NWD,NWD,"Q",KLIST)
18710= 12   IF(NWDD.EQ.0) GO TO 18
18720=      IF(IFLTR.LE.0) GO TO 13
18730=      WRITE 106,NWDD
18740= 106  FORMAT(" ENTER DISTURBANCE NOISE STRENGTHS: ",I2)
18750=      CALL RDWGTS(DM(LQN),NWDD,0)
18760= 13   CALL DVCTOR(NWDD,DM(LQN),ZM1)
18770=      CALL MATLST(ZM1,NWDD,1,"QN",KTERM)
18780= 15   CALL MATLST(DM(LQN),NWDD,NWDD,"QN",KLIST)
18790= 18   CALL QDSCRT(DM(LQ),DM(LQN),ZM1,ZM2)
18800=      IF(IFLTR.LE.0) GO TO 19
18810=      WRITE 107,NMD
18820= 107  FORMAT(" ENTER MEASUREMENT NOISE STRENGTHS: ",I2)
18830=      CALL RDWGTS(DM(LR),NMD,0)
18840= 19   CALL DVCTOR(NMD,DM(LR),ZM1)
18850=      CALL MATLST(ZM1,NMD,1,"R",KTERM)
18860= 20   CALL MATLST(DM(LR),NMD,NMD,"R",KLIST)
18870= 25   CALL TFRMTX(DM(LHP),SM,NMD,NDIM,2)
18880=      CALL TRANS2(NMD,NDIM,SM,ZM1)
18890=      LFCOV=LFLTRK+NDIM*NMD
18900=      CALL DVCTOR(NMD,DM(LR),FLT(LFCOV))
18910=      CALL KFLTR(NDIM,NMD,FLT,ZM1,DM(LQD),FLT(LFCOV),ZM2,
18920=     1 FLT(LFLTRK),SM)
18930=      CALL TFRMTX(SM,COM2,NDIM,NDIM,2)
18940=      IA=1
18950=      DO 30 I=1,NPLD
18960=      FLT(LFCOV-1+I)=SQRT(ZM2(IA))
18970= 30   IA=IA+NDIM1
```

```
      CALL MATLST(FLT(LFLTRK),NDIM,NMD,"KF",KLIST)
      CALL MATLST(FLT(LFLTRK),NDIM,NMD,"KF",KTERM)
      IFLTR=1
      LFLKF=1
  111 FORMAT(A3)
      RETURN
C END SUBROUTINE FLTRK
      END
      SUBROUTINE FEVAL
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/MAIN2/COM2(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CXF/NVFLT,FLT(1)
      DIMENSION ITITLE(5),NS(3),NVPLOT(2)
      IF(NWT.GT.0) GO TO 1
      WRITE(KTERM,100)
  100 FORMAT("0NO TRUTH MODEL DRIVING NOISE - - FILTER EVALUATION ABORTE
     1D")
      RETURN
    1 WRITE(KLIST,110)
  110 FORMAT(////11X,5("* "),"FILTER EVALUATION",5(" *")////)
      CALL FMMUL(COM2,FLT(LEADSN),NPLD,NPLD,NPLD,SM)
      CALL POLES(SM,NPLD,5,ZM1,ZM2)
      NA=NNT+NPLD
      NSIZE=NA*NA
      IF(NSIZE.LE.NVSM) GO TO 8
      WRITE 101,NSIZE
  101 FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
      GO TO 9
    8 IF(NSIZE.LE.NVZM) GO TO 10
      WRITE 103,NSIZE
  103 FORMAT("0INSUFFICIENT MEMORY /ZMTX1/,/ZMTX2/, NEED: ",I4)
    9 LABORT=NSIZE
      RETURN
   10 CALL ZPART(SM,1,NSIZE,1)
      NDIM=NPLD
      NDIM1=NDIM+1
      CALL TFRMTX(TM(LRT),ZM1,NMD,NMD,2)
      CALL MAT3(NPLD,NMD,FLT(LFLTRK),ZM1,COM1)
      NVOUT=2*NPLD+1
      REWIND KPLOT
      CALL DACOV(SM,FLT(LFCOV),ZM1,ZM2,NA,NVOUT,0.)
      DO 20 IT=1,50
      TIME=TSAMP*FLOAT(IT)
      CALL ACOVUD(SM,TM(LQDT),COM1,TM(LPHT),FLT(LEADSN),
     1 COM2,ZM1,ZM2)
      CALL DACOV(SM,FLT(LFCOV),ZM1,ZM2,NA,NVOUT,TIME)
   20 CONTINUE
      ENDFILE KPLOT
      WRITE(KTERM,104)
      READ(KIN,102) ITITLE
      DO 50 I=1,NPLD
      REWIND KPLOT
      NS(1)=1
      NS(2)=52
      NS(3)=103
      NVPLOT(1)=I+I-1
      NVPLOT(2)=I+I
      DO 40 J=1,51
      CALL RPLOTF(ZM1,NVOUT,IERR)
      IF(IERR.EQ.1) GO TO 50
      CALL STRPLT(SM,ZM1,NS,NVPLOT,2,NVOUT)
      DO 35 K=1,3
   35 NS(K)=NS(K)+1
   40 CONTINUE
      WRITE 107,ZM1(NVPLOT(1)),I,ZM1(NVPLOT(2))
  107 FORMAT("0FINAL RMS ERRORS : TRUE = ",1PE15.7/"  (STATE",I3,
     1 ")",4X,"COMPUTED = ",1PE15.7)
      CALL PLOTLP(51,2,SM,-1,1,0,KLIST,ITITLE)
      WRITE(KLIST,106) I
  106 FORMAT("0   STATE : ",I2//4X,"SYMBOL 1 : TRUE ERROR"/
     1 4X,"SYMBOL 2 : COMPUTED ERROR "/)
   50 CONTINUE
      RETURN
  104 FORMAT(" +",10("-")," ENTER TITLE IN GIVEN FIELD ",10("-"),"+"/)
  102 FORMAT(5A10)
C END SUBROUTINE FEVAL
      END
      SUBROUTINE DACOV(PCA,PC,ZM1,ZM2,NA,NVOUT,TIME)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      DIMENSION PCA(1),PC(1),ZM1(1),ZM2(1)
      NDIM=NA
      NDIM1=NDIM+1
      CALL TFRMTX(TM(LTDT),ZM1,NWD,NNT,2)
      IF(NDD.LT.1) GO TO 5
      IA=LADDR(NA,NND+1,1)
      CALL TFRMTX(TM(LTNT),ZM1(IA),NDD,NNT,2)
    5 CALL MSCALE(ZM1,ZM1,NPLD,NNT,-1.)
      IA=LADDR(NA,1,NNT+1)
      CALL IDNT(NPLD,ZM1(IA),1.)
      CALL MAT3(NPLD,NA,ZM1,PCA,ZM2)
      WRITE(KLIST,101) TIME
  101 FORMAT('0'TRUE' DESIGN ERROR COVARIANCE AT TIME = ',F6.4)
      CALL MATIO(ZM2,NPLD,NPLD,3)
      IA=1
      DO 10 I=1,NPLD
      NS=I+I
      ZM1(NS-1)=SQRT(ZM2(IA))
      ZM1(NS)=PC(I)
   10 IA=IA+NDIM1
      ZM1(NVOUT)=TIME
      CALL WPLOTF(ZM1,NVOUT)
      RETURN
C END SUBROUTINE DACOV
      END
      SUBROUTINE ACOVUD(PC,QD,RKRKT,PHIT,PHI,RINKH,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,CON1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NWPR
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LDCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      DIMENSION PC(1),QD(1),RKRKT(1),PHIT(1),PHI(1),RINKH(1),
     1 ZM1(1),ZM2(1)
      L1=LADDR(NDIM,1,NNT+1)
      CALL ZPART(ZM2(L1),NNT,NPLD,NDIM)
      CALL TFRMTX(PHIT,ZM2,NNT,NNT,2)
      L1=LADDR(NDIM,NNT+1,NNT+1)
      CALL TFRMTX(PHI,ZM2(L1),NPLD,NPLD,2)
      L2=LADDR(NDIM,NNT+1,1)
      CALL ZPART(ZM2(L2),NPLD,NNT,NDIM)
      CALL MAT3(NDIM,NDIM,ZM2,PC,ZM1)
      CALL FPADD(ZM1,NDIM,QD,NNT,NNT,1,PC)
      CALL IDNT(NNT,ZM2,1.)
      CALL FMMUL(FLT(LFLTRK),TM(LHT),NPLD,NMD,NNT,ZM1)
      CALL TFRMTX(ZM1,ZM2(L2),NPLD,NNT,2)
      CALL TFRMTX(RINKH,ZM2(L1),NPLD,NPLD,2)
      CALL MAT3(NDIM,NDIM,ZM2,PC,ZM1)
      CALL FPADD(ZM1,NDIM,RKRKT,NPLD,NPLD,L1,PC)
      RETURN
C END SUBROUTINE ACOVUD
      END
      SUBROUTINE FPADD(X,NX,Y,NRY,NCY,LADDR,Z)
      DIMENSION X(1),Z(1),Y(NRY,NCY)
      CALL FTMTX(X,Z,NX,NX)
      LAM1=LADDR-1
      DO 10 I=1,NCY
      LA1=LAM1+NX*(I-1)
      DO 10 J=1,NRY
      LA1=LA1+1
   10 Z(LA1)=Z(LA1)+Y(J,I)
      RETURN
C END SUBROUTINE FPADD
      END
      SUBROUTINE RSYS(A,L,ND,ITYPE,IWRT)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/AMC/AM(1)
      COMMON/BDG/BD(1)
      DIMENSION A(1),L(1),ND(1),NAD(14,2),IND(7,3),NTYP(2,3),NTITLE(3),
     1 NMAT(14,3)
      DATA NTYP/7,14,3,4,4,8/
      DATA NO/1HN/
      DATA IND/1HN,1HR,1HP,1HM,1HD,1HW,2HWD,2HNM,2HRM,2HPM,4(1H ),
     1 2HNT,2HRT,2HMT,2HWT,1H ,1H ,1H /
      DATA NTITLE/6HDESIGN,7HCOMMAND,5HTRUTH/
      DATA NMAT/1HA,1HB,2HEX,1HG,1HQ,1HC,2HDY,2HEY,1HH,2HHN,1HR,2HA N,
     1 2HGN,2HQN,2HAN,2HBN,2HCN,2HDN,10(1H ),2HAT,2HBT,2HGT,2HQT,2HHT,
     2 2HRT,3HTDT,3HTNT,6(1H )/
      NDM=NTYP(1,ITYPE)
      NAR=NTYP(2,ITYPE)
      NT=NTITLE(ITYPE)
      WRITE(KLIST,110) NT
  110 FORMAT(////11X,5("* "),A7," MODEL",5(" *")////)
    5 WRITE 101,NT
  101 FORMAT("0READ ",A7," MODEL FROM 'DATA' FILE (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 10
      IF=1
      CALL READFS(A,ND,ITYPE,IERR)
      IF(IERR.EQ.0) GO TO 201
   10 WRITE 102,NT
  102 FORMAT(" ENTER ",A7," MODEL FROM TERMINAL (Y OR N) >")
      READ 111,IANS
      IF(IANS.EQ.NO) GO TO 15
      IF=2
      DO 12 I=1,NDM
      WRITE 112,IND(I,ITYPE)
  112 FORMAT(" ENTER ",A2," >")
   12 READ*,ND(I)
      GO TO 201
   15 IF=3
      IF(ITYPE-2) 16,17,18
   16 CALL DSND(ND)
      GO TO 20
   17 CALL CMDD(ND)
      GO TO 20
   18 CALL TRTHD(ND)
   20 IF(ND(1).GT.0) GO TO 201
      WRITE 114,NT
  114 FORMAT("0",A7," MODEL SUBROUTINE NON-EXISTENT")
      GO TO 5
  201 IF(ITYPE-2) 21,22,23
   21 CALL DSNDM(ND,NAD)
      GO TO 25
   22 CALL CMDDM(ND,NAD)
      GO TO 25
```

```
21140= 23   CALL TRTHDM(ND,NAD)
21150= 25   IF(LABORT.EQ.0) GO TO 26
21160=      WRITE 103,NT,LABORT
21170= 103  FORMAT("0INSUFFICIENT MEMORY FOR ",A7," MODEL, NEED: ",I4)
21180=      RETURN
21190= 26   L(1)=1
21200=      DO 30 I=2,NAR
21210= 30   L(I)=L(I-1)+NAD(I-1,1)*NAD(I-1,2)
21220=      NPNTS=L(NAR)+NAD(NAR,1)*NAD(NAR,2)-1
21230=      IF(NPNTS.LE.NVSM) GO TO 34
21240=      WRITE 104,NPNTS
21250= 104  FORMAT("0INSUFFICIENT MEMORY /SYSMTX/, NEED: ",I4)
21260=      LABORT=NPNTS
21270=      RETURN
21280= 34   IF(IF-2) 75,35,50
21290= 35   IZ=1
21300=      DO 40 I=1,NAR
21310=      N1=NAD(I,1)
21320=      N2=NAD(I,2)
21330=      IF((N1.EQ.0).OR.(N2.EQ.0)) GO TO 40
21340=      WRITE 113,NMAT(I,ITYPE)
21350= 113  FORMAT("0ENTER ",A3)
21360=      CALL ZMATIN(A(L(I)),N1,N2,IZ)
21370= 40   CONTINUE
21380=      GO TO 75
21390= 50   CALL ZPART(A,1,NPNTS,1)
21400=      IF(ITYPE-2) 55,60,65
21410= 55   CALL DSNM(A(L(1)),A(L(2)),A(L(3)),A(L(4)),A(L(5)),A(L(6)),A(L(7)),
21420=     1 A(L(8)),A(L(9)),A(L(10)),A(L(11)),A(L(12)),A(L(13)),A(L(14)))
21430=      GO TO 75
21440= 60   CALL CNDM(A(L(1)),A(L(2)),A(L(3)),A(L(4)))
21450=      GO TO 75
21460= 65   CALL TRTHM(A(L(1)),A(L(2)),A(L(3)),A(L(4)),A(L(5)),A(L(6)),
21470=     1 A(L(7)),A(L(8)))
21480= 75   IZ=0
21490= 77   WRITE 105
21500= 105  FORMAT("0MODIFY MATRIX ELEMENTS (Y OR N) >")
21510=      READ 111,IANS
21520=      IF(IANS.EQ.NO) GO TO 88
21530=      WRITE 106,(NMAT(I,ITYPE),I=1,NAR)
21540= 106  FORMAT(1X,14(2X,A3))
21550= 78   WRITE 107
21560= 107  FORMAT(" ENTER MATRIX NAME >")
21570=      READ 111,IANS
21580=      DO 80 I=1,NAR
21590=      IF(IANS.EQ.NMAT(I,ITYPE)) GO TO 81
21600= 80   CONTINUE
21610=      GO TO 78
21620= 81   WRITE 116
21630= 116  FORMAT(" LIST MATRIX TO TERMINAL (Y OR N) >")
21640=      READ 111,IANS
21650=      IF(IANS.EQ.NO) GO TO 83
21660=      CALL MATLST(A(L(I)),NAD(I,1),NAD(I,2),NMAT(I,ITYPE),KTERM)
21670= 83   CALL ZMATIN(A(L(I)),NAD(I,1),NAD(I,2),IZ)
```

```
      GO TO 77
 88   IF(ITYPE.EQ.2) CALL FTMTX(A(L(1)),AM,ND(1),ND(1))
      IF(ITYPE.EQ.1) CALL FTMTX(A(L(2)),BD,ND(1),ND(2))
 90   IF(IWRT) 95,92,93
 92   IWRT=1
 93   WRITE 115,NT
 115  FORMAT("0WRITE ",A7," MODEL TO 'SAVE' FILE (Y OR N) >")
      READ 111,IANS
/-----------deletion------------/
      IF(IANS.EQ.NO) GO TO 95
      CALL WFILED(ITYPE,NPNTS,ND,A)
      IWRT=-1
      WRITE 109,NT
 109  FORMAT(6X,A7," MODEL WRITTEN TO 'SAVE' FILE")
 95   DO 100 I=1,NAR
      N1=NAD(I,1)
      N2=NAD(I,2)
      IF((N1.EQ.0).OR.(N2.EQ.0)) GO TO 100
      CALL MATLST(A(L(I)),N1,N2,NMAT(I,ITYPE),KLIST)
 100  CONTINUE
 111  FORMAT(A3)
      RETURN
C END SUBROUTINE RSYS
      END
      SUBROUTINE DSND(ND)
      DIMENSION ND(1)
      ND(1)=0
      RETURN
C END SUBROUTINE DSND
      END
      SUBROUTINE CNDD(ND)
      DIMENSION ND(1)
      ND(1)=0
      RETURN
C END SUBROUTINE CNDD
      END
      SUBROUTINE TRTHD(ND)
      DIMENSION ND(1)
      ND(1)=0
      RETURN
C END SUBROUTINE TRTHD
      END
      SUBROUTINE DSNM(A,B,EX,G,Q,C,DY,EY,H,HD,R,AD,GD,QD)
      RETURN
C END SUBROUTINE DSNM
      END
      SUBROUTINE CNDM(AM,BM,CM,DM)
      RETURN
C END SUBROUTINE CNDM
      END
      SUBROUTINE TRTHM(AT,BT,GT,QT,HT,RT,TDT,TNT)
      RETURN
C END SUBROUTINE TRTHM
      END
```

```
22210=      SUBROUTINE DSNDM(ND,NAD)
22220=      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
22230=      COMMON/NDIND/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
22240=      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
22250=      DIMENSION ND(1),NAD(14,2)
22260=      NND=ND(1)
22270=      NRD=ND(2)
22280=      NPD=ND(3)
22290=      NMD=ND(4)
22300=      NDD=ND(5)
22310=      NWD=ND(6)
22320=      NWDD=ND(7)
22330=      NPLD=NND+NDD
22340=      NWPNWD=NWD+NWDD
22350=      NNPR=NND+NRD
22360=      NAD(1,1)=NND
22370=      NAD(2,1)=NND
22380=      NAD(3,1)=NND
22390=      NAD(4,1)=NND
22400=      NAD(5,1)=NWD
22410=      NAD(6,1)=NPD
22420=      NAD(7,1)=NPD
22430=      NAD(8,1)=NPD
22440=      NAD(9,1)=NMD
22450=      NAD(10,1)=NMD
22460=      NAD(11,1)=NMD
22470=      NAD(12,1)=NDD
22480=      NAD(13,1)=NDD
22490=      NAD(14,1)=NWDD
22500=      NAD(1,2)=NND
22510=      NAD(2,2)=NRD
22520=      NAD(3,2)=NDD
22530=      NAD(4,2)=NWD
22540=      NAD(5,2)=NWD
22550=      NAD(6,2)=NND
22560=      NAD(7,2)=NRD
22570=      NAD(8,2)=NDD
22580=      NAD(9,2)=NND
22590=      NAD(10,2)=NDD
22600=      NAD(11,2)=NMD
22610=      NAD(12,2)=NDD
22620=      NAD(13,2)=NWDD
22630=      NAD(14,2)=NWDD
22640=      NSIZE=NPLD*(2*NPLD+NND+NMD+NPD+NWPNWD)+NRD*(NND+NPD)+
22650=     1 NDD*NDD+NMD*NMD+NWD*NWD+NWDD*NWDD
22660=      IF(NSIZE.GT.NVDM) LABORT=NSIZE
22670=      RETURN
22680=C END SUBROUTINE DSNDM
22690=      END
22700=      SUBROUTINE CMDDM(ND,NAD)
22710=      COMMON/NDIMC/NNC,NRC,NPC
22720=      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
22730=      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
22740=      DIMENSION ND(1),NAD(14,2)
```

```
      NNC=ND(1)
      NRC=ND(2)
      NPC=ND(3)
      NAD(1,1)=NNC
      NAD(2,1)=NNC
      NAD(3,1)=NPC
      NAD(4,1)=NPC
      NAD(1,2)=NNC
      NAD(2,2)=NRC
      NAD(3,2)=NNC
      NAD(4,2)=NRC
      NSIZE=NNC*(NNC+NRC+NPC)+NPC*NRC
      IF(NSIZE.GT.NVCM) LABORT=NSIZE
      RETURN
C END SUBROUTINE CMDDM
      END
      SUBROUTINE TRTHDM(ND,NAD)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDINT/NNT,NRT,NMT,NWT
      COMMON/TRUMTX/NVTM,TM(1)
      DIMENSION ND(1),NAD(14,2)
      NNT=ND(1)
      NRT=ND(2)
      NMT=ND(3)
      NWT=ND(4)
      NAD(1,1)=NNT
      NAD(2,1)=NWT
      NAD(3,1)=NNT
      NAD(4,1)=NWT
      NAD(5,1)=NMT
      NAD(6,1)=NMT
      NAD(7,1)=NWD
      NAD(8,1)=NDD
      NAD(1,2)=NNT
      NAD(2,2)=NRT
      NAD(3,2)=NWT
      NAD(4,2)=NWT
      NAD(5,2)=NNT
      NAD(6,2)=NMT
      NAD(7,2)=NNT
      NAD(8,2)=NNT
      NSIZE=NNT*(2*NNT+NMD+NRD+NPLD)+NMD*NMD
      IF(NSIZE.GT.NVTM) LABORT=NSIZE
      RETURN
C END SUBROUTINE TRTHDM
      END
      SUBROUTINE ZMATIN(A,NR,NC,IZ)
      DIMENSION A(NR,NC)
      IF(IZ) 10,10,1
 1    DO 5 I=1,NR
      DO 5 J=1,NC
 5    A(I,J)=0.
 10   WRITE 101,NR,NC
```

```
   15 READ*,I,J,V
      IF(I.EQ.0) RETURN
      IF((I.LE.NR).AND.(J.LE.NC)) GO TO 20
      WRITE 102
      GO TO 10
   20 A(I,J)=V
      IF(IZ.LT.0) A(J,I)=V
      GO TO 15
  101 FORMAT(" ENTER I,J AND M(I,J)--(0/ WHEN COMPLETE) : "I2," BY "I2,
     1 " )")
  102 FORMAT(" ERROR IN ARRAY INDEX")
C END SUBROUTINE ZMATIN
      END
      SUBROUTINE WFILED(NT,NP,ND,A)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION ND(15),A(NP)
      DATA IEOI/-1/
      BACKSPACE KSAVE
      WRITE(KSAVE,101) NT,NP
      WRITE(KSAVE,101) (ND(I),I=1,15)
      WRITE(KSAVE,102) (A(I),I=1,NP)
      WRITE(KSAVE,101) IEOI,NP
      RETURN
  101 FORMAT(2I4)
  102 FORMAT(E20.10)
C END SUBROUTINE WFILED
      END
      SUBROUTINE READFS(A,ND,NT,IERR)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION A(1),ND(15)
      DATA IEOI/-1/
      REWIND KDATA
    5 READ(KDATA,102) IT,NP
      IF(IT.NE.IEOI) GO TO 10
      WRITE 101
      IERR=1
      RETURN
   10 CALL FARRAY(A,ND,NP)
      IF(IT.NE.NT) GO TO 5
      IERR=0
  101 FORMAT("0DATA NOT IN 'DATA' FILE . . .")
  102 FORMAT(2I4)
      RETURN
C END SUBROUTINE READFS
      END
      SUBROUTINE FARRAY(A,ND,NP)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION A(NP),ND(15)
      READ(KDATA,101) (ND(I),I=1,15)
      READ(KDATA,102) (A(I),I=1,NP)
      RETURN
  101 FORMAT(2I4)
  102 FORMAT(E20.10)
C END SUBROUTINE FARRAY
```

```
      END
      SUBROUTINE TFRNTX(X1,X2,NR,NC,ITX)
      COMMON/MAIN1/NDIM
      DIMENSION X1(1),X2(1)
      IF(ITX.EQ.2) GO TO 20
      J=NC*NDIM
      KK=0
      DO 10 I=1,J,NDIM
      L=I+NR-1
      DO 10 JJ=I,L
      KK=KK+1
   10 X1(KK)=X2(JJ)
      RETURN
   20 KK=NR*NC+1
      DO 30 I=1,NC
      L=(NC-I)*NDIM+1
      DO 30 J=1,NR
      KK=KK-1
      JJ=L+NR-J
   30 X2(JJ)=X1(KK)
      RETURN
      END
      SUBROUTINE MATLST(A,NR,NC,NT,KDEV)
      DIMENSION A(NR,NC)
      WRITE(KDEV,101) NT
      DO 10 I=1,NR
   10 WRITE(KDEV,102) (A(I,J),J=1,NC)
  101 FORMAT(1H0,A3," MATRIX"/)
  102 FORMAT(1X1P10G13.4)
      RETURN
C END SUBROUTINE MATLST
      END
      SUBROUTINE NDSCRT(A,N,NTERMS)
      COMMON/DESIGN/NVCON,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      DIMENSION A(1)
      NTERMS=IFIX(6.+3.*TSAMP*XNORM(N,A))
      IF(NTERMS.GT.30) NTERMS=30
      RETURN
C END SUBROUTINE NDSCRT
      END
      SUBROUTINE RQWGTS(W,ND,NP)
      DIMENSION W(1)
   10 WRITE 101
  101 FORMAT(" ENTER I AND QW(I,I)--(0/ WHEN COMPLETE) >")
   15 READ*,I,V
      IF(I.EQ.0) RETURN
      IF(I.LE.ND) GO TO 20
      WRITE 102
  102 FORMAT(" ERROR IN ARRAY INDEX")
      GO TO 10
   20 IF(V) 25,30,40
   25 WRITE 103
  103 FORMAT(" ELEMENTS MUST BE NON-NEGATIVE")
      GO TO 15
```

```
 30   IF(NP) 35,40,35
 35   WRITE 104
 104  FORMAT(" ELEMENTS MUST BE POSITIVE")
      GO TO 15
 40   L1=LADDR(ND,I,I)
      W(L1)=V
      GO TO 15
C END SUBROUTINE RDWGTS
      END
      SUBROUTINE DVCTOR(N,A,V)
      DIMENSION A(1),V(1)
      NP1=N+1
      N2=N*N
      J=0
      DO 10 I=1,N2,NP1
      J=J+1
 10   V(J)=A(I)
      RETURN
C END SUBROUTINE DVCTOR
      END
      SUBROUTINE POLES(A,N,ITYPE,ZM1,ZM2)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION NTYP(5),A(1),ZM1(1),ZM2(1)
      DATA NTYP/6HDESIGN,7HCOMMAND,5HTRUTH,5HREGPI,6HFILTER/
      NDS=NDIM
      NDIM=N
      NDIM1=NDIM+1
      CALL EIGEN(NDIM,A,ZM1,ZM1(NDIM1),ZM2,0)
      IF(ITYPE.LT.4) GO TO 10
      CALL MAPOLE(N,ZM1,ZM1(NDIM1),TSAMP)
 10   WRITE(KLIST,102) NTYP(ITYPE)
      WRITE(KTERM,102) NTYP(ITYPE)
      WRITE(KLIST,101) (ZM1(I),ZM1(NDIM+I),I=1,N)
      WRITE(KTERM,101) (ZM1(I),ZM1(NDIM+I),I=1,N)
      NDIM=NDS
      NDIM1=NDIM+1
 101  FORMAT(6X,1PE15.7,"  +J(",1PE15.7,")")
 102  FORMAT("0POLES OF ",A7," MATRIX"/)
      RETURN
C END SUBROUTINE POLES
      END
      SUBROUTINE MAPOLE(N,ZR,ZI,T)
      DIMENSION ZR(1),ZI(1)
      RT=1./T
      DO 10 I=1,N
      ZM=SQRT(ZR(I)**2+ZI(I)**2)
      SIGMA=RT*ALOG(ZM)
      ZI(I)=RT*ATAN2(ZI(I),ZR(I))
 10   ZR(I)=SIGMA
      RETURN
C END SUBROUTINE MAPOLE
      END
      FUNCTION LADDR(NR,I,J)
      LADDR=I+NR*(J-1)
      RETURN
C END FUNCTION LADDR
      END
      SUBROUTINE FTMTX(X,Y,NR,NC)
      DIMENSION X(1),Y(1)
      NE=NR*NC
      DO 10 I=1,NE
 10   Y(I)=X(I)
      RETURN
C END SUBROUTINE FTMTX
      END
      SUBROUTINE FMMUL(X,Y,NR1,NC1,NC2,Z)
      DIMENSION X(NR1,NC1),Y(NC1,NC2),Z(NR1,NC2)
      DOUBLE PRECISION TD
      DO 10 I=1,NR1
      DO 10 J=1,NC2
      TD=0.D00
      DO 5 K=1,NC1
 5    TD=TD+X(I,K)*Y(K,J)
 10   Z(I,J)=TD
      RETURN
C END SUBROUTINE FMMUL
      END
      SUBROUTINE FTMUL(X,Y,NR1,NC1,NC2,Z)
      DIMENSION X(NR1,NC1),Y(NR1,NC2),Z(NC1,NC2)
      DOUBLE PRECISION TD
      DO 10 I=1,NC1
      DO 10 J=1,NC2
      TD=0.D00
      DO 5 K=1,NR1
 5    TD=TD+X(K,I)*Y(K,J)
 10   Z(I,J)=TD
      RETURN
C END SUBROUTINE FTMUL
      END
      SUBROUTINE FMADD(X,Y,NR,NC,Z)
      DIMENSION X(1),Y(1),Z(1)
      NE=NR*NC
      DO 10 I=1,NE
 10   Z(I)=X(I)+Y(I)
      RETURN
C END SUBROUTINE FMADD
      END
      SUBROUTINE ZPART(A,NR,NC,ND)
      DIMENSION A(1)
      NE=NC*ND
      DO 10 I=1,NR
      DO 10 J=I,NE,ND
 10   A(J)=0.
      RETURN
C END SUBROUTINE ZPART
      END
      SUBROUTINE SUBI(A,NR,ND)
      DIMENSION A(1)
      ND1=ND+1
      NE=NR*ND
      DO 10 I=1,NE,ND1
 10   A(I)=A(I)-1.
      RETURN
C END SUBROUTINE SUBI
      END
      SUBROUTINE WPLOTF(V,N)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION V(N)
      WRITE(KPLOT,101) (V(I),I=1,N)
      RETURN
 101  FORMAT(E20.10)
C END SUBROUTINE WPLOTF
      END
      SUBROUTINE RPLOTF(V,N,IERR)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      DIMENSION V(N)
      READ(KPLOT,101) (V(I),I=1,N)
      IF(EOF(KPLOT)) 5,10
 5    IERR=1
      RETURN
 10   IERR=0
      RETURN
 101  FORMAT(E20.10)
C END SUBROUTINE RPLOTF
      END
      SUBROUTINE STRPLT(A,V,NS,NV,NP,NVO)
      DIMENSION A(1),NS(1),NV(1),V(1)
      A(NS(1))=V(NVO)
      DO 5 I=1,NP
 5    A(NS(I+1))=V(NV(I))
      RETURN
C END SUBROUTINE STRPLT
      END
      SUBROUTINE PLOTLP(N,M,A,IPSC,ISCL,LPTERM,NDEV,ITITLE)
C * * * * * * * * * *
C * N = NUMBER OF POINTS TO BE PLOTTED
C * M = NUMBER OF OUTPUTS TO BE PLOTTED
C * A = VECTOR OF SAMPLE POINTS FOR PLOTTING : DIMENSION = N*M
C *     ELEMENTS 1 TO N ARE THE INDEPENDENT VARIABLE
C *     ELEMENTS (N+1) TO 2*N, (2*N+1) TO 3*N, AND SO ON ARE
C *     THE DEPENDENT VARIABLES--EACH VARIABLE IS IN CONSECUTIVE
C *     STORAGE WITH CORRESPONDING SAMPLE POINTS FOR EACH
C *     SEPARATED BY MULTIPLES OF N
C * IPSC = -1 => SCALE ALL VARIABLES TOGETHER (1 PLOT)
C *      =  0 => SCALE TOGETHER AND SEPARATELY (2 PLOTS)
C *      = +1 => SCALE SEPARATELY (1 PLOT)
C * ISCL = 0 => PLOT OVER EXACT RANGE OF VARIABLE
C *      = 1 => PLOT USING EVEN SCALING
C * LPTERM = 0 => PLOT IS TO TERMINAL (50 CHARACTERS WIDE)
C *        = 1 => PLOT IS TO LINE PRINTER (100 CHARACTERS WIDE)
```

```
25990=C * NDEV = DEVICE NUMBER FOR PLOT OUTPUT
26000=C * ITITLE = VECTOR(DIMENSIONED 5) WITH 50 CHARACTER TITLE
26010=C * * * * * * * * * * *
26020=      DIMENSION YSCAL(6),YMIN(6),IBLNK(6),YPR(11),A(1)
26030=      INTEGER OUT(101),SYMBOL(6),BLANK,PLUS,GRID,ITITLE(5)
26040=      DATA BLANK,PLUS,COLON,SYMBOL/1H ,1H+,1H:,1H1,1H2,1H3,1H4,1H5,1H6/
26050= 1    FORMAT(1H )
26060= 2    FORMAT(1H1,11X,5A10/)
26070= 10   FORMAT(1H ,F11.2,6X,101A1)
26080= 12   FORMAT(11H0    SCALE ,A1,1X,11F10.4)
26090=      IPAPER=5*(1+LPTERM)
26100=      ISPAC=10*IPAPER
26110=      RISPAC=FLOAT(ISPAC)
26120=      ISPAC=ISPAC+1
26130=      IPRT1=IPAPER+1
26140=      RMIN=A(N+1)
26150=      RMAX=RMIN
26160= 25   DO 41 ISC=1,M
26170=      M1=ISC*N+1
26180=      YL=A(M1)
26190=      YH=YL
26200=      M2=N*(ISC+1)
26210=      DO 40 J=M1,M2
26220=      IF(A(J).LT.YL) GO TO 30
26230=      IF(A(J).GT.YH) YH=A(J)
26240=      GO TO 40
26250= 30   YL=A(J)
26260= 40   CONTINUE
26270=      IF(YL.LT.RMIN)RMIN=YL
26280=      IF(YH.GT.RMAX) RMAX=YH
26290=      IF(IPSC.GE.0) CALL VARSCL(YL,YH,YSCAL(ISC),RISPAC,ISCL)
26300= 41   YMIN(ISC)=YL
26310=      IF(IPSC.LE.0)CALL VARSCL(RMIN,RMAX,SCAL,RISPAC,ISCL)
26320=      IC=2-IABS(IPSC)
26330=      DO 45 IX=1,ISPAC
26340= 45   OUT(IX)=BLANK
26350=      DO 100 ICO=1,IC
26360=      WRITE(NDEV,2) (ITITLE(I),I=1,5)
26370=      DO 60 I=1,N
26380=      XPR=A(I)
26390=      IF(MOD(I,10).EQ.0) GO TO 458
26400=      GRID=BLANK
26410=      GO TO 460
26420= 458  GRID=COLON
26430= 460  DO 461 IX=2,ISPAC,2
26440= 461  OUT(IX)=GRID
26450=      DO 46 IX=1,ISPAC,10
26460= 46   OUT(IX)=PLUS
26470=      DO 55 J=1,M
26480=      IL=I+J*N
26490=      IF(IPSC) 48,47,49
26500= 47   IPSCT=IPSC+ICO
26510=      IF(IPSCT.EQ.2) GO TO 49
26520= 48   JP=IFIX((A(IL)-RMIN)/SCAL)+1
```

```
      GO TO 50
 49   JP=IFIX((A(IL)-YMIN(J))/YSCAL(J))+1
 50   OUT(JP)=SYMBOL(J)
 55   IBLNK(J)=JP
      WRITE(NDEV,10) XPR,(OUT(IX),IX=1,ISPAC)
      DO 59 J=1,M
      ITEMP=IBLNK(J)
 59   OUT(ITEMP)=BLANK
 60   CONTINUE
      IF(IPSC) 68,67,72
 67   IF(IPSCT.EQ.2) GO TO 72
 68   YPR(1)=RMIN
      DO 70 I=1,IPAPER
 70   YPR(I+1)=YPR(I)+10.*SCAL
      WRITE(NDEV,12) BLANK,(YPR(I),I=1,IPRT1)
      GO TO 100
 72   DO 76 ISC=1,M
      YPR(1)=YMIN(ISC)
      DO 74 I=1,IPAPER
 74   YPR(I+1)=YPR(I)+10.*YSCAL(ISC)
 76   WRITE(NDEV,12) SYMBOL(ISC),(YPR(IX),IX=1,IPRT1)
 100  WRITE(NDEV,1)
      RETURN
C END SUBROUTINE PLOTLP
      END
      SUBROUTINE VARSCL(XMIN,XMAX,SCALE,RSPACE,ISCL)
      IF(XMAX.EQ.XMIN) XMIN=.9*XMIN-10.
      SCALE=XMAX-XMIN
      IF(ISCL.EQ.0) GO TO 25
      EXP=IFIX(100.+ALOG10(SCALE))-100.
      FACTOR=10.**(1.-EXP)
      XMINT=XMIN*FACTOR
      XMAXT=XMAX*FACTOR
      IF(XMAXT.GE.0.) XMAXT=XMAXT+.9
      IF(XMINT.LE.0.) XMINT=XMINT-.9
      XMINT=AINT(XMINT)
      ISCAL=XMAXT-XMINT
      IF(MOD(ISCAL,5).NE.0) ISCAL=ISCAL+5-MOD(ISCAL,5)
      FACTOR=10.**(EXP-1.)
      XMIN=XMINT*FACTOR
      SCALE=FACTOR*FLOAT(ISCAL)
 25   SCALE=SCALE/RSPACE
      RETURN
C END SUBROUTINE VARSCL
      END
C
C*********************************************************************
C
C CODE FROM THIS POINT ON WAS ADDED BY LT MOSELEY (1982)
C
C*********************************************************************
C
      SUBROUTINE PFDATA(ICODE,ND)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/MAIN2/COM2(1)
      COMMON/INOU/KIN,KOUT,KPUNCH
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/ZMTX2/ZM2(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NWPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LGN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/LOCC/LPHC,LBDC,LCC,LDC
      COMMON/CMDMTX/NVCM,NEWCM,NODC,CM(1)
      COMMON/NDIMT/NNT,NRT,NMT,NWT
      COMMON/LOCT/LPHT,LBDT,LQDT,LHT,LRT,LTDT,LTNT
      COMMON/TRUMTX/NVTM,TM(1)
      COMMON/LCNTRL/LPI11,LPI12,LPI21,LPI22,LPHDL,LBDL
      COMMON/CONTROL/NVCTL,CTL(1)
      COMMON/LREGPI/LXDW,LUDW,LPHCL,LKX,LKZ
      COMMON/CREGPI/NVRPI,RPI(1)
      COMMON/LCGT/LA11,LA13,LA21,LA23,LA12,LA22,LKXA11,LKXA12,LKXA13
      COMMON/CCGT/NVCGT,CGT(1)
      COMMON/LKF/LEADSN,LFLTRK,LFCOV
      COMMON/CKF/NVFLT,FLT(1)
      DIMENSION ND(1)
      ND(1)=NND
      ND(2)=NRD
      ND(3)=NPD
      ND(4)=NMD
      ND(5)=NDD
      ND(6)=NPLD
      ND(7)=NNC
      ND(8)=NRC
      ND(9)=NPC
      ND(10)=NNT
      ND(11)=NRT
      ND(12)=NMT
      ND(13)=NODY
      ND(14)=NOEY
      NVZMS=NVZM
      CALL FTMTX(DM(LPHI),SM,NND,NND)
      LL=NND*NND+1
      CALL FTMTX(DM(LBD),SM(LL),NND,NRD)
      LL=NND*NRD+LL
      IF(NDD.EQ.0) GO TO 100
         CALL FTMTX(DM(LEX),SM(LL),NND,NDD)
         LL=NND*NDD+LL
         CALL FTMTX(DM(LPHD),SM(LL),NDD,NDD)
         LL=NDD*NDD+LL
         IF(NODY.EQ.1) GO TO 90
            CALL FTMTX(DM(LDY),SM(LL),NPD,NRD)
            LL=NPD*NRD+LL
 90      IF(NOEY.EQ.1) GO TO 95
            CALL FTMTX(DM(LEY),SM(LL),NPD,NDD)
```

```
                 LL=NPD*NDD+LL
 95         CALL FTMTX(DM(LHP),SM(LL),NMD,NPLD)
            LL=NMD*NPLD+LL
            GO TO 200
 100  CONTINUE
      IF(NODY.EQ.1) GO TO 105
            CALL FTMTX(DM(LDY),SM(LL),NPD,NRD)
            LL=NPD*NRD+LL
 105  CALL FTMTX(DM(LHP),SM(LL),NMD,NMD)
      LL=NMD*NMD+LL
 200  CALL FTMTX(DM(LC),SM(LL),NPD,NMD)
      LL=NPD*NMD+LL
      CALL FTMTX(CM(LPHC),SM(LL),NNC,NNC)
      LL=NNC*NNC+LL
      CALL FTMTX(CM(LBDC),SM(LL),NNC,NRC)
      LL=NNC*NRC+LL
      CALL FTMTX(CM(LCC),SM(LL),NPC,NNC)
      LL=NPC*NNC+LL
      CALL FTMTX(TM(LPHT),SM(LL),NNT,NNT)
      LL=NNT*NNT+LL
      CALL FTMTX(TM(LBDT),SM(LL),NNT,NRT)
      LL=NNT*NRT+LL
      CALL FTMTX(TM(LQDT),SM(LL),NNT,NNT)
      LL=NNT*NNT+LL
      CALL FTMTX(TM(LHT),SM(LL),NMT,NNT)
      LL=NMT*NNT+LL
      CALL FTMTX(TM(LRT),SM(LL),NMT,NMT)
      LL=NMT*NMT+LL
      CALL FTMTX(RPI(LKX),SM(LL),NRD,NMD)
      LL=NRD*NMD+LL
      CALL FTMTX(RPI(LKZ),SM(LL),NRD,NPD)
      LL=NRD*NPD+LL
      CALL FTMTX(CGT(LKXA11),SM(LL),NRC,NNC)
      LL=NRC*NNC+LL
      IF(NDD.EQ.0) GO TO 300
            CALL FTMTX(CGT(LKXA13),SM(LL),NRD,NDD)
            LL=NRD*NDD+LL
            CALL FTMTX(FLT(LFLTRK),SM(LL),NPLD,NMD)
            LL=NPLD*NMD+LL
            CALL FTMTX(TM(LTDT),SM(LL),NMD,NNT)
            LL=NMD*NNT+LL
            CALL FTMTX(TM(LTNT),SM(LL),NDD,NNT)
            LL=NDD*NNT+LL
            GO TO 310
 300  CONTINUE
      CALL FTMTX(FLT(LFLTRK),SM(LL),NND,NMD)
      LL=NMD*NMD+LL
      CALL FTMTX(TM(LTDT),SM(LL),NND,NNT)
      LL=NMD*NNT+LL
 310  SM(LL)=TSAMP
      ND(15)=LL
      CALL WFILED(ICODE,LL,ND,SM)
      NVZM=NVZMS
      RETURN
```

```
C END SUBROUTINE PFDATA
      END
      SUBROUTINE IMPLEX(A)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/DESIGN/NVCOM,TSAMP,LFLRPI,LFLCGT,LFLKF,LTEVAL,LABORT
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/ZMTX1/NVZM,ZM1(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NWPR
      COMMON/LOCD/LAP,LGP,LPHI,LBD,LEX,LPHD,LQ,LQN,LQD,LC,LDY,LEY,LHP,LR
      COMMON/DSNMTX/NVDM,NODY,NOEY,DM(1)
      COMMON/NDIMC/NNC,NRC,NPC
      COMMON/AMC/AM(1)
      COMMON/BDG/BD(1)
      DIMENSION A(1)
      DATA NO/1HN/
/-----------deletion-------------/
  215 WRITE(KLIST,126)
  126 FORMAT(5("* "),"COMBINED IMPLICIT/EXPLICIT CONTROL",5(" *"))
/-----------deletion-------------/
      LQI=1
      LRII=LQI+NPD*NPD
      LRRI=LRII+NRD*NRD
      NPDNRD=NPD+NRD
  120 CALL ZPART(A,NPDNRD,NPDNRD,NPDNRD)
      WRITE 122,NPD
  122 FORMAT(" ENTER WEIGHTS ON IMPLICIT OUTPUT DERIVATIVES: ",I2)
      CALL RQWGTS(A,NPD,0)
      WRITE 124,NRD
  124 FORMAT(" ENTER WEIGHTS ON IMPLICIT CONTROL MAGNITUDES: ",I2)
      CALL RQWGTS(A(LRII),NRD,1)
      L1=1
      L2=L1+NND*NND
      L3=L2+NPD*NND
      L4=L3+NPD*NND
      L5=L4+NPD*NND
      L6=L5+NND*NPD
      L7=L6+NPD*NND
      IF(NDD.EQ.0) GO TO 5
           NDIM=NPLD
           NDIM1=NDIM+1
           GO TO 10
    5 NDIM=NND
      NDIM1=NDIM+1
   10 CALL TFRMTX(SM(L1),DM(LAP),NND,NND,1)
      CALL FMMUL(DM(LC),SM(L1),NPD,NND,NND,SM(L2))
      CALL FMMUL(AM,DM(LC),NPD,NPD,NND,SM(L3))
      NDIM=NPD
      NDIM1=NDIM+1
      CALL MADD1(NPD,NND,SM(L2),SM(L3),SM(L4),-1.)
      CALL FTRNSP(NPD,NND,SM(L4),SM(L5))
      CALL FMMUL(A,SM(L4),NPD,NPD,NND,SM(L6))
      CALL FMMUL(SM(L5),SM(L6),NND,NPD,NND,SM(L1))
      CALL FTRNSP(NND,NRD,BD,SM(L2))
      CALL FTRNSP(NPD,NND,DM(LC),SM(L3))
      CALL FMMUL(SM(L2),SM(L3),NRD,NND,NPD,SM(L4))
      CALL FMMUL(SM(L4),SM(L6),NRD,NPD,NND,SM(L5))
      CALL FTMTX(SM(L5),SM(L2),NRD,NND)
      CALL FMMUL(SM(L4),A,NRD,NPD,NPD,SM(L5))
      CALL FMMUL(SM(L5),DM(LC),NRD,NPD,NND,SM(L6))
      CALL FMMUL(SM(L6),BD,NRD,NND,NRD,SM(L7))
      CALL MADD1(NRD,NRD,A(LRII),SM(L7),SM(L5),1.)
      CALL FTMTX(SM(L5),SM(L3),NRD,NRD)
      WRITE 400
 400  FORMAT("0LIST QIH MATRIX TO TERMINAL (Y OR N) >")
      READ 410,IANS
 410  FORMAT(A3)
      IF(IANS.EQ.NO) GO TO 490
         CALL MATLST(SM(L1),NND,NND,"QIH",KTERM)
         WRITE 420
 420     FORMAT("0CHANGE IMPLICIT WEIGHTS (Y OR N) >")
         READ 410,IANS
         IF(IANS.EQ.NO) GO TO 490
         GO TO 120
 490  CALL MATLST(A,NPD,NPD,"QI",KLIST)
      CALL DVCTOR(NPD,A,A(LRRI))
      CALL MATLST(A(LRRI),NPD,1,"QI",KTERM)
      CALL MATLST(A(LRII),NRD,NRD,"RI",KLIST)
      CALL DVCTOR(NRD,A(LRII),A(LRRI))
      CALL MATLST(A(LRRI),NRD,1,"RI",KTERM)
      CALL MATLST(SM(L1),NND,NND,"QIH",KLIST)
      CALL MATLST(SM(L2),NRD,NND,"SIH",KLIST)
      CALL MATLST(SM(L3),NRD,NRD,"RIH",KLIST)
      RETURN
C END  SUBROUTINE IMPLEX
      END
      SUBROUTINE MODIFX(A)
      COMMON/MAIN1/NDIM,NDIM1,COM1(1)
      COMMON/FILES/KSAVE,KDATA,KPLOT,KLIST,KTERM
      COMMON/SYSMTX/NVSM,SM(1)
      COMMON/NDIMD/NND,NRD,NPD,NMD,NDD,NWD,NWDD,NPLD,NWPNWD,NNPR
      COMMON/NDIMC/NNC,NRC,NPC
      DIMENSION A(1)
      L1=1
      L2=L1+NND*NND
      L3=L2+NRD*NND
      L4=L3+NRD*NRD
      L5=L4+NND*NND
      L6=L5+NND*NND
      NDIM=NNPR
      NDIM1=NDIM+1
      CALL TFRMTX(SM(L4),A,NND,NND,1)
      CALL FMADD(SM(L4),SM,NND,NND,SM(L5))
      CALL TFRMTX(SM(L5),A,NND,NND,2)
      LLL=LADDR(NDIM,NND+1,1)
      CALL TFRMTX(SM(L4),A(LLL),NRD,NND,1)
      CALL FMADD(SM(L4),SM(L2),NRD,NND,SM(L5))
      CALL TFRMTX(SM(L5),A(LLL),NRD,NND,2)
      CALL FTRNSP(NRD,NND,SM(L5),SM(L6))
      LLL=LADDR(NDIM,1,NND+1)
      CALL TFRMTX(SM(L6),A(LLL),NND,NRD,2)
      LLL=LADDR(NDIM,NND+1,NND+1)
      CALL TFRMTX(SM(L4),A(LLL),NRD,NRD,1)
      CALL FMADD(SM(L4),SM(L3),NRD,NRD,SM(L5))
      CALL TFRMTX(SM(L5),A(LLL),NRD,NRD,2)
      CALL MATLST(A,NNPR,NNPR,"XIE",KLIST)
      RETURN
C END SUBROUTINE MODIFX
      END
      SUBROUTINE FTRNSP(NR,NC,A,B)
      DIMENSION A(1),B(1)
      DO 10 I=1,NR
         DO 20 J=1,NC
               B((I-1)*NC+J)=A((J-1)*NR+I)
 20   CONTINUE
 10   CONTINUE
      RETURN
C END SUBROUTINE FTRNSP
      END
```

D.5 CGTPIV Segmentation Job Control

CGTPIB = Binary object code

CGTPIV = Executable load module

DLKLIB = Library routines [16,23]

```
1ØØ=WGM. T83Ø283,MILLER
11Ø=MAP,FULL.
12Ø=ATTACH,CGTPIB.
13Ø=ATTACH,DLKLIB,ID=T82Ø3Ø3.
14Ø=LIBRARY,DLKLIB.
15Ø=REQUEST,CGTPIV,*PF.
16Ø=SEGLOAD(B=CGTPIV)
17Ø=LOAD(CGTPIB)
18Ø=NOGO.
19Ø=REWIND,CGTPIV.
2ØØ=CATALOG,CGTPIV,CGTPIV,RP=18Ø.
21Ø=*EOR
22Ø=SETUP      INCLUDE DSCRT
23Ø=SREGPI     INCLUDE RQWGTS,MLINEQ,SYMFCT
24Ø=FLTRK      INCLUDE RQWGTS,KFLTR,MLINEQ,SYMFCT,INTEG
25Ø=STRTH      INCLUDE DSCRTT,INTEG
26Ø=SDSN       INCLUDE QDSCRT
27Ø=CEVAL      INCLUDE PLOTLP,VARSCL,RPLOTF,WPLOTF,STRPLT
28Ø=FEVAL      INCLUDE PLOTLP,VARSCL,RPLOTF,WPLOTF,STRPLT,DACOV
29Ø=B1         TREE SETUP-(SDSN,SCMD,STRTH)
3ØØ=B2         TREE PIMTX
31Ø=B3         TREE SREGPI
32Ø=B4         TREE SCGT
33Ø=B5         TREE CEVAL
34Ø=B6         TREE FLTRK
35Ø=B7         TREE FEVAL
36Ø=B8         TREE PFDATA
37Ø=A          TREE CGTXQ-(B1,B2,B3,B4,B5,B6,B7,B8)
38Ø=ROOT       TREE MAIN-A
39Ø=           GLOBAL MAIN1,MAIN2,INOU,DESIGN,FILES,SYSMTX,ZMTX1,ZMTX2,
4ØØ=,NDIMD,LOCD,DSNMTX,NDIMC,LOCC,CMDMTX,NDIMT,LOCT,TRUMTX,LCNTRL,CONTROL,
41Ø=,LREGPI,CREGPI,LCGT,CCGT,LKF,CKF,AMC,BDG
42Ø=         END
```

E. Discussion of Design Software Error

E.1 Introduction

After the designs for this study had been developed and analyzed and, in fact, after this report had been written, an error was found in the original CGT/PI/KF design software. This error, in turn, had become a part of the program version (CGTPIV) which was used for the designs in this research. The impact of that error on the results and conclusions presented in this report was evaluated by using the corrected software to reaccomplish a cross-section of the design work. While the impact was assessed to be significant, it did not invalidate the main findings presented in this thesis. The time available to complete this project was far less than that which would have been required to reaccomplish all of its original objectives in a thorough manner, using the corrected software. Under the circumstances, it was felt that documentation of the error and its impact, within this report, would be more practical and more beneficial than attempting to rewrite the report in its entirety without adequate data. The bulk of this thesis was therefore left intact, in order to avoid discarding the good (i.e., the theoretical development, the design approach and the use of analysis tools) with the bad (primarily, incorrect observations concerning the type and extent of the differences between designs achievable through the standard PI regulator design method as compared to the implicit model-following formulation). Clearly, the designs presented in Chapter V of this report are "good" designs, as was demonstrated by performance analysis. Appropriate gains were found despite the software error. The significance of the error was that it made the design process more difficult, and restricted the choice of design paths which were fruitful.

This appendix discusses the error that was found and its impact on the designs and conclusions presented in the body of this report. Revised insights as to the differences between the standard and implicit model-following regulator formulations are provided. A few sample regulator designs are presented to demonstrate the use of the corrected software. These designs are preliminary in nature, and have not been refined to the extent that those presented in Chapter V were. They do, however, point out the differences observed due to the software correction and demonstrate the validity of the work previously presented.

E.2 The Software Error

Appendix A reviewed the formulation of a constant-gain, LQ, optimal PI regulator, and related the development to the CGT/PI/KF design software. In using that software, the designer specifies a number of continuous-time quadratic weighting factors which become a part of a cost function that is mathematically minimized to determine the optimal gains for the controller. As one might imagine, a great deal of mathematical manipulation takes place between the entry of the quadratic weights at the terminal and the final revelation of the desired gains. The process is explained in detail in Volume 2 of [16]. It includes discretization of the continuous-time cost function, transformation to remove cross-weighting terms so that a standard Riccati equation solver can be used, solution of the matrix Riccati equation, and retransformation of the resulting gains for use in the original system coordinates. During this complex process, the array containing the discretized weighting factors for the control rates was used, in the original program, as the input argument of "GMINV," a library subroutine [23] which calculates a matrix inverse:

```
10440=       CALL GMINV(NRD,NRD,U,ZM1,MR,1)
```

The above line of code calls "GMINV" to find the inverse (or pseudoinverse, if appropriate) of the "NRD" by "NRD" matrix "U," returning the result as matrix "ZM1." "MR" is the calculated rank of the inverse, and the "1" is an input flag to enable the printing of error messages. The matrix "U" is, however, destroyed in the computation of "ZM1." In the original program, the destroyed matrix was inadvertently used in a subsequent calculation, thus producing the incorrect controller gains.

To correct the problem, line 10440, above, was replaced by

```
10435=       CALL EQUATE(PHIP,U,NRD,NRD)
10440=       CALL GMINV(NRD,NRD,PHIP,ZM1,MR,1)
```

The first line calls "EQUATE" to copy the "NRD" by "NRD" matrix "U" into the array "PHIP." "PHIP" is then used as the calling argument for "GMINV," thus preserving "U". This change has been incorporated into the source listing given in Section D.4, and, in fact, the given line reference numbers apply to that listing.

Because of the complexity of the calculations just described, the error was well disguised to the novice LQ designer. As long as the values of the weights on the control rates were kept small in relation to the weights placed on the control magnitudes and on the outputs (or output rates), the effect of the invalid matrix on the computed gain did not generally prevent iteratively converging on a design with good performance. Several erroneous impressions arose early in the study because of the relationship just described. Very small increases in the weights on the control rates were seen to be effective in slowing down the initial response of the regulator (which was an objective in the design due to the rate-limited actuators). But larger increases (to more than about 3, given the values of the other weights being used at the time) had been found to cause overshoots and oscillations in the actuators and outputs. Also, only a limited amount of regulator speed control had been readily achievable with the standard regulator. Therefore, the extensive further efforts at slowing the regulator were directed to the design path using implicit model-following, with various regulator command models and varying weights on the output rates.

The locations of the closed-loop regulator poles were calculated independently, and not affected by the error; they were correct for the quadratic weights assigned but, unfortunately, wrong with respect to the actual gains which defined the controllers. Thus, the prediction of robustness with respect to higher-order dynamics models in Section 5.6, while based on sound theory and, in fact, effective due to the relative pole locations for the designs, was actually conducted on the basis of the wrong absolute pole locations. This condition gave rise to an additional misconception -- that the use of implicit model-following could provide much greater and more systematic control over the locations of the closed-loop system poles than could the standard formulation. This appeared to be the case because the erroneous software seemingly allowed a greater range of quadratic weights to be placed on the implicit model-following output rates than on the outputs of the standard regulator. The differences in the reported pole locations were greater than the amount of change actually occurring in the true system poles due to the difference in gain.

The gains that were calculated were incorrect only with respect to the quadratic weights assigned by the designer. The designs produced were valid, since the controller is defined by its gains and the control law in which the gains are used. The gains were "good" because they were shown to produce good performance in simulation analysis. Very similar designs were achieved with the corrected software, and simply occurred with different quadratic weights, which were the result of design iteration.

E.3 The Impact of the Error on the Design Results

To demonstrate the effect of the correction to the software on the design process, consider the baseline standard PI regulator (SR-B) discussed in Chapter V. The quadratic weights used in conducting the design with the original software are listed in Table V-1, as are the gains calculated by the program. Using the same quadratic weights with the corrected program produced the following regulator gains:

$$\underline{K}_x = \begin{bmatrix} -181.5 & 36.97 & -6.79 & 3.147 & 0.3260 \\ 124.8 & -153.2 & -4.25E-2 & 0.3115 & 1.795 \end{bmatrix} \qquad \text{(E-1a)}$$

and

$$\underline{K}_z = \begin{bmatrix} -4.699 & -2.86 \\ -4.272 & 9.92 \end{bmatrix} \qquad \text{(E-1b)}$$

These gains differed significantly from those given for SR-B in Table V-1, and so did the performance of the controller. The reported closed-loop pole locations were, however, the same as given for SR-B in Table V-3. Compared to that of SR-B, summarized in Table V-2 and shown in Figure V-1, the transient response for this regulator using the linear single-state actuator model was extremely fast, with a rise time of 0.06 seconds, but with better damping (an overshoot of only 20% in the pitch angle). Accordingly, the horizontal tail and trailing edge flap control actuators were driven to well over twice their true position limits and at initial rates of about 50 and 20 times their respective rate limits (given in Section 4.3). Considering the poor results with much slower regulators discussed in Chapter V, there was not much point in simulating the performance of this regulator with the nonlinear actuator model which included rate and position limits. When the regulator's response to initial conditions was evaluated with respect to the linear three- and four-state actuator models, the design was grossly unstable, with state magnitudes reaching values on the order of 10^{18} within 2 seconds of the start of the simulation, whereas SR-B had been oscillatory, but stable, under the same conditions.

By decreasing the weights on the outputs (the "Y" matrix of CGTPIV) from 200 to 5 each, the value of $\underline{X}(3,3)$ in the state weighting matrix from 50 to 2, and also increasing the weights on the control rates (the "UR" matrix) from 1 to 10 each, a design with gains and performance much more like the original baseline regulator was achieved. The pole locations for such a design should, therefore, be reasonably similar to the true pole locations for SR-B. For the design just described, the poles were located at:

$-13.6 \pm j15.5$
$-1.87 \pm j0.07$
-27.8
-15.9
-20.0

which were considerably different from the erroneous poles listed for SR-B in Table V-3. With the corrected software, the values of the output weightings could be reduced even further relative to the input rate weightings, producing designs very similar to those achieved through implicit model-following with a two state command model; it is important to note that this had not been possible with the uncorrected software. At the same time, the greater range of changes in the quadratic weightings in the standard regulator designs also permitted systematic control of the location of the closed-loop regulator poles, as had earlier been thought possible only when implicit model-following was employed. Examples of these observation will follow in the next section.

The effect on the design process with implicit-model following paralleled that just outlined for the standard regulator. Use of the quadratic weightings of the designs from Chapter V resulted in high gains, large actuator commands, and gross instability with respect to higher-order dynamics. Designs with gains and performance characteristics similar to those evaluated in Chapter V where achievable through reduction of the output rate weightings (the "QI" matrix of CGTPIV) and by increasing the weights on the control rates (the "UR" matrix). With the corrected software, representing the actuator states as outputs of the implicit regulator command model appeared to present an additional feasible design option; the excessive initial actuator commands which had occurred when this was attempted prior to the software change were found to be countered by using increased control rate weightings.

In both standard and implicit designs, increasing the weight on the control magnitudes ("UM" or "RI" matrices) did not yield a reduction in initial control surface actuation; it generally still caused the initial control inputs to be increased, as had been observed in Section 4.4.3.

In summary, the primary impact of the software error on the design results of this study was as follows:

1. The designs presented in the main body of the thesis were valid, as were the analyses conducted using simulation and singular values. Achieving the designs with the incorrect software resulted from the use of quadratic weightings which would have been inappropriate when used with the corrected software. The quadratic weightings used in the research were the result of a systematic, iterative design process which would have, in fact, been easier had the design software functioned properly.

2. With the corrected software, the standard regulator formulation was capable of producing designs similar to those based on implicit model-following with a two-state command model. Use of the standard formulation still required designer insight for the manual alteration of the (3,3) element of the state weighting matrix ("X") to improve the output damping [16]; the implicit model-following formulation provided the proper weighting automatically for a wide range of designs. This, in fact, was the primary difference between the design of the standard regulator versus that of the implicit regulator based on the two-state command model. With the corrected

software, the use of higher-order regulator command models still appeared to provide a relatively greater degree of designer control over some aspects of the design characteristics; specifically, the use of a three-state command model allowed the regulator to be slowed to a much greater extent without loss of proper damping, alleviating the need for designer insight to modify the structure of the state weighting matrix.

3. The locations for the closed-loop regulator poles given by the erroneous software did not correspond to the correct locations associated with the actual controller gains. However, the relative motion of the poles given, in response to changes in the quadratic weightings, was correct. The example analyses in the sequel will demonstrate that the use of the correct regulator pole locations in predicting the robustness of the designs in the face of neglected higher-order dynamics provides an even greater degree of insight than had been proposed using the erroneous absolute pole locations in Chapter V.

4. The calculation of the CGT gains was not affected by the error in the software, nor were any aspects of the singular value robustness analysis for the PI regulator designs.

With these facts in mind, the reader who has not yet conducted an in-depth coverage of Chapters IV, V and VI is encouraged to do so prior to reading the next section.

E.4 Analysis of New Example Designs

Several example preliminary designs which support the conclusions of the preceding section were assembled and analyzed in a manner similar to that of Chapter V. Time was not available to explore the possibilities for improving on these initial designs by further experimentation with quadratic weights or different regulator command models, but the designs presented do provide an indication of the characteristics of the designs achievable with the corrected software, and allow the analysis methods previously applied to be exercised and validated. From this point on, it is assumed that the reader is thoroughly familiar with the design results and analysis methodology of Chapter V. The presentation here is brief, and the analysis results are reviewed only in enough depth to characterize the performance and robustness of the "new" designs. Most of the controllers discussed are very similar to those presented in the main body of this thesis; they were simply achieved using different quadratic weights. Refer to Tables E-1 through E-6 at the end of this appendix. These tables present data in a format similar to that used in Chapter V; this data will be used to summarize the controller characteristics. Time response plots are not included, as they were deemed to provide little additional information or insight. The names given the new regulators parallel those in Chapter V, with a "C" added to indicate "Corrected software." CGT designs are not discussed, as the CGT gain calculation, given a set of applicable regulator gains, was not effected by the software change.

Designs SR-1C and SR-2C were based on the standard PI regulator formulation. As seen in Table E-1, lower weights on the outputs and greater weights on input rates were used than had been employed for the standard regulator designs of Chapter V (referring to Table V-1). SR-1C exhibited transient performance similar to that of SR-B (Table V-2) in response to initial conditions, as outlined in Table E-3; it was slightly faster, as characterized by rise and peak times, but more heavily damped. The initial inputs to the actuators were also similar to those of SR-B, and the maximum stable initial condition with respect to the nonlinear actuators was therefore nearly the same. When tested against the four-state linear actuators, SR-1C was highly unstable (output values on the order of 10^6 in 2 seconds), whereas SR-B had been stable, but oscillatory. When tested at the .6 mach/20,000 feet flight condition, its performance was similar to that of SR-B, which was quite good. The minimum singular values for the inverse return difference function as well as closed-loop regulator pole locations are given in Table E-5, and will be referred to in subsequent comparisons.

Design SR-2C was based on even lower output weights and higher input rate weights, and thus provided a slower transient response than the previous design; similar, in fact, to the IMF2 class regulators of Section 5.4. Thus, it displayed a predictably larger tolerance than did SR-1C for initial conditions using nonlinear actuators. It also performed well against the linear four-state actuator model. This might have been predicted by its relatively larger minimum singular values over the range of 10 to 100 radians per second, and by the location of the complex regulator poles -- closer to the origin and much further from the perturbing actuator poles. Because it had a larger overall minimum singular value than did SR-1C, its somewhat worse performance at .6 mach and 20,000 feet would not have been predicted by singular value analysis.

Referring again to the tables, IMF2-1C was the first "new" implicit model-follower, with performance similar to that of the old IMF2-1. Comparison of Tables V-4 and E-1 thus provides insight as to the amount of adjustment required in the quadratic weights due to the software change. IMF2-1C's stable, but oscillatory, performance against the linear four-state actuator model could have been predicted by either singular value or pole-location analysis, in comparison to the values and performance of the standard regulators just reviewed. Singular value analysis once again would have provided little insight into the performance of IMF2-1C at .6 mach and 20,000 feet; its minimum singular value was about the same as that of SR-2C, but its performance was worse than that of either of the two previous designs.

IMF2-2C was based on a slower regulator command model and larger input rate weights, partially offset by larger output rate weights than in the previous design. These weights were not purposefully chosen by comparison to those of the previous design, but resulted from a series of iterations about a different design point. The design had a slower rise time, better damping, and a better capability with respect to nonlinear (saturating) actuators, in comparison to IMF2-1C. Note, from Table E-5, that the only poles that moved significantly (compared to IMF2-1C) were the two nearest the origin, which generally track the command model poles. Since the complex poles did not move appreciably, and in consideration of the similarity of their minimum singular values, it is not surprising that the performance of the two implicit model-followers with respect to the higher-order actuators was nearly the same. Again, singular value analysis provided little insight regarding performance at .6 mach and 20,000 feet.

IMF2-3C differed from the previous design only because of reduced weights on the output rates. Its rise time therefore increased and so did its capability with regard to actuator saturation. Pole-location and singular value analysis would have predicted the nearly identical performance with the linear four-state actuators to that of SR-2C. Note, in fact, that all of the entries in each of the tables are nearly the same for the two designs -- including the regulator gains. Nearly identical designs were produced with the two different regulator design formulations; the primary difference between the two methods, so far as the designer is concerned, would have been the insight required to modify the (3,3) element of the standard formulation's "X" matrix to provide the same damping as that provided automatically by the implicit formulation (the actual value of $\underline{X}_{IE}(3,3)$ in the combined state weighting matrices for both designs was precisely 1.0).

IMF3-1C used the "slow" three-state regulator command model. As was the case with the IMF3 designs of Section 5.5, this resulted in very slow transient response, but very good damping. However, the capability with respect to actuator saturation was less than might have been expected for such a slow regulator, and additional iterations would have been necessary to resolve that problem. A more serious problem with the design was the vulnerable position of its complex poles which, in conjunction with the poor singular values shown in Table E-6, adequately predicted that the controller would be unstable when tested with the linear four-state actuators. As usual, singular value analysis would not have predicted this controller's relatively good performance at .6 mach and 20,000 feet.

IMF3-2C was designed with the specific goal of "safely" placing the complex regulator poles. The faster command model was used, since it had been observed to affect primarily only the poles nearest the origin on the real axis, so as to avoid slowing the transient response unnecessarily. As shown in Table E-2, however, radical adjustments were made in the output rate and input rate weightings to drive the complex poles toward the origin. The effort was successful, as shown in Table E-6; both pole locations and singular values predicted the excellent performance achieved with the four state actuators. Although probably somewhat slow for use in a fighter flight control system (indicating the need for additional design iteration), this regulator was also capable of handling a 7 degree initial condition with nonlinear actuators and performed reasonably well at .6 mach and 20,000 feet. Although the results are not shown in any of the tables, a hybrid CGT/PI using this regulator in conjunction with the CGT gains of IMF2-5 also provided reasonably good performance in following a step command input.

The final design shown in the tables was somewhat experimental. A four-state regulator command model was used, with outputs to be tracked by the pitch angle, flight path angle, and the two actuator states. This had been tried with the uncorrected software and, although the rise time had been slow, the initial commands to the actuators had always been excessive due to the distribution of weights in the "RIH" matrix, as discussed in Section 4.4.3. With the corrected software, this problem was overcome by the small output rate weightings and the large input rate weightings, producing a transient response similar to that achieved by earlier IMF3 designs. Note, in Table E-6, the pole location given at -254.4 ±j157.1; this is the condition noted in Section 4.4.5, wherein a pole lies on the negative real axis of the z-plane and cannot correctly be mapped into the s-plane. The value 157.1 is pi times the controller sampling rate. The effect of this pole on the controller characteristics was not known until it was tested against the linear four-state actuator model and found to be highly unstable, with output magnitudes on the order of 10^3 after 2 seconds. Again, this would have been predicted by the small high frequency singular values for the controller. On the other hand, the large overall minimum singular value for this design predicted possible good robustness features with respect to parameter variations. When evaluated at .6 mach and 20,000 feet, IMF4-1C exhibited a much slower rise time (0.72 seconds) than at design conditions and very low frequency oscillations, with an apparent settling time of about 3 seconds. This change in response rate was unique among all of the designs presented. Although not without problems, the design has some interesting characteristics, and merits further research. Very little work was done in this study with such a four-state command model due to problems generated by the design software error. The analysis of this design was presented only to show that the correction of the software provided even more avenues for exploration regarding implicit model-following design.

E.5 Summary

This appendix documented the error discovered in the design software subsequent to the completion of the research work presented in the rest of this report. The primary direct impact of the error on the validity of the results presented in this thesis was shown to be the incorrect correlation between the values used for design quadratic weights and the gains of the PI regulators. A lack of correspondence also occurred between the given closed-loop regulator pole locations and the calculated gains. The two effects combined to produce misconceptions about the design process and appropriate design strategies. The limitations imposed by the error resulted in a large portion of the design effort being directed at finding ways to control the speed of the initial regulator response.

With the corrected software, the speed of the regulator response could be controlled effectively by changing the relative values of the output (or output rate) weightings and those of the control rates. With the standard regulator formulation, the designer was required to modify the structure of the state weighting matrix to increase the response damping. This was not required, when using implicit model-following with a two-state command model, until the weights on the control rates exceeded those on the output rates by several orders of magnitude. When good damping was required with even slower response, it could easily be achieved through the use of the three-state command model.

If the design problem discussed in this report were to be reconsidered in the light of the above insights, an appropriate design approach using the corrected software would be similar to the one that was followed in this study. With the software providing more logical and predictable results due to the correction, the design process should produce the desired performance with fewer iterations. The systematic design process, using implicit model-following, could be summarized as follows:

1. Develop alternative regulator designs of the IMF2 and IMF3 (and, perhaps, IMF4 and IMF5) classes, basing the initial design iterations on the need to achieve adequate transient response to initial conditions despite the nonlinear (saturating) actuators. The primary means of reducing the speed of the initial response of the regulator would be by increasing the magnitude of the weights on the control rates (the "UR" matrix) relative to the magnitude of the weights on the output rates (the "QI" matrix). Since only the relative sizes of the various weights are of significance, the simplest policy would be to keep the weights on the control magnitudes (the "RI" matrix) constant, such as an identity matrix, and vary only "UR" and "QI." Designs based on several different regulator command models of each class should be included to provide a variety of different closed-loop characteristics for further analysis.

2. Analyze the alternative designs with respect to higher-order actuator dynamics models. Use pole-location and singular value analysis in conjunction with simulation results to identify favorable robustness trends that result from varied quadratic weighting and command model combinations.

3. Test the designs which perform adequately with both nonlinear and higher-order actuator dynamics models for robustness with respect to parameter variations, through simulation with perturbed dynamics

models.

4. Design CGT/PI controllers based on the regulators which, in the light of the previous tests, are best suited for implementation. For regulators based on command models of dimension greater than the number of control inputs (2 in this case), the use of a hybrid CGT/PI (as discussed in Section 5.5) would probably be required. This is due to the inadequacy of the CGT design formulation when faced with a rank defective $\underline{\Pi}$ matrix, as discussed in Section 4.4.7. Analyze the CGT/PI designs for acceptable tracking of command inputs, and to ensure that the CGT control inputs do not destablize the overall system when operating on nonlinear (saturating) actuators.

5. At any point in this process, trend information that arises from the analysis results can be used to "retune" a regulator design, or series of designs, in an effort to improve controller performance. In the final implementation, the use of anti-windup compensation, as discussed in Section 5.7, should be considered as a means of enhancing stability characteristics without unnecessarily degrading performance.

The validity of the design approach and use of analysis tools presented in the body of this report was not adversely affected by the software error. The same approach and tools worked just as well, if not better, with the initial designs based on the corrected software. Had this study been conducted using the corrected software, a great deal more about the true capabilities and characteristics of implicit model-following might have been learned; it now remains a topic for future research. However, except as already noted, the conclusions reached in this research effort remain basically unchanged.

Table E-1. Definition of New Sample Designs (Part 1)

Design	Quadratic Weights	Command Model	Regulator Gains						
			$\underline{K}_x$					$\underline{K}_z$	
SR-1C	$\underline{Y}$=diag(10,10) $\underline{U}_M$=diag(1,1) $\underline{U}_R$=diag(5,5) $\underline{X}$(3,3)=5	N/A	-53.15 65.47	17.51 -74.42	-2.872 6.68E-2	1.867 0.2148	0.2185 0.9678	-0.4564 -2.659	-0.7983 2.723
SR-2C	$\underline{Y}$=diag(5,5) $\underline{U}_M$=diag(1,1) $\underline{U}_R$=diag(20,20) $\underline{X}$(3,3)=1	N/A	-21.77 37.43	9.646 -40.75	-1.437 6.9E-2	1.142 0.1438	0.1458 0.5381	-0.1162 -1.47	-0.4258 1.436
IMF2-1C	$\underline{Q}_I$=diag(1,1) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(10,10)	P=5	-34.85 53.58	14.51 -57.87	-1.956 0.1482	1.429 0.1515	0.1587 0.7372	-0.8226 -2.047	-0.7776 2.221
IMF2-2C	$\underline{Q}_I$=diag(3,3) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(20,20)	P=2	-29.27 40.06	10.35 -43.81	-1.852 3.5E-2	1.357 0.1767	0.1849 0.5705	-0.4 -1.442	-0.3998 1.529

Table E-2. Definition of New Sample Designs (Part 2)

Design	Quadratic Weights	Command Model	Regulator Gains	
			$\underline{K}_x$	$\underline{K}_z$
IMF2-3C	$\underline{Q}_I$=diag(1,1) $\underline{R}_I$=diag(1,1) $\underline{U}_R$=diag(20,20)	P=2	-21.88 9.604 -1.45 1.15 0.1455 38.53 -41.6 8.2E-2 0.1416 0.5446	-0.1235 -0.3618 -1.453 1.438
IMF3-1C	$\underline{Q}_I$=diag(5,5,.01) $\underline{R}_I$=diag(1,1,1) $\underline{U}_R$=diag(10,10,10)	P_1=5 P_2=5 P_3=10	-7.1E-2 20.86 -4.066 1.957 0.2388 5.5E-2 -79.94 0.8699 0.2912 0.9058	-3.875 -1.841 -1.391 -2.256 5.853 1.498
IMF3-2C	$\underline{Q}_I$=diag(.1,.1,.001) $\underline{R}_I$=diag(1,1,1) $\underline{U}_R$=diag(20,20,20)	P_1=10 P_2=10 P_3=15	-2.5E-2 10.2 -1.58 1.087 0.1286 2.85E-2 -41.1 0.4797 0.1619 0.5449	-0.2789 -0.498 -0.4237 -1.499 1.479 0.764
IMF4-1C	$\underline{Q}_I$=diag(1,1,.1,.1) $\underline{R}_I$=diag(1,1,1,1) $\underline{U}_R$=diag(10,10,10,10)	P_1=10 P_2=10 P_3=20 P_4=20	-11.29 -10.94 3.5071 4.508 0.3074 -2.439 -2.374 0.7299 0.3188 3.121	-2.75 0.75 3.31 0.48 -0.55 0.30 0.49 1.11

Table E-3. Summary of Performance Analysis for New Sample Designs (Part 1)

Condition and Measure of Performance	SR-1C	SR-2C	IMF2-1C	IMF2-2C
Percentage of overshoot in pitch angle (truth model=linear design model)	20	35	46	30
Rise time/Peak time (seconds) for pitch angle (truth model=linear design model)	0.10/0.20	0.16/0.30	0.12/0.24	0.14/0.24
Maximum pitch angle initial condition (degrees) for stability (nonlinear actuators*)	0.4	1.1	0.5	0.7
Percentage of overshoot in pitch angle (truth model=linear 4th-order actuators*)	Unstable	50	90	70
Percentage of initial condition present in oscillation at 2 seconds (truth model=linear 4th-order actuators*)	Unstable	None	30	20
Percentage of overshoot in pitch angle (truth model= .6 mach/20,000 feet)	90	110	120	100
Percent of initial condition present in oscillation at 2 seconds (truth model= .6 mach/20,000 feet)	None	3	20	2

*Truth model is otherwise identical to linear design model

Table E-4. Summary of Performance Analysis for New Sample Designs (Part 2)

Condition and Measure of Performance	IMF2-3C	IMF3-1C	IMF3-2C	IMF4-1C
Percentage of overshoot in pitch angle (truth model=linear design model)	30	None	10	20
Rise time/Peak time (seconds) for pitch angle (truth model=linear design model)	0.16/0.30	0.32/--	0.4/0.8	0.28/0.56
Maximum pitch angle initial condition (degrees) for stability (nonlinear actuators*)	1.1	1.1	7.0	2.0
Percentage of overshoot in pitch angle (truth model=linear 4th-order actuators*)	45	Unstable	10	Unstable
Percentage of initial condition present in oscillation at 2 seconds (truth model=linear 4th-order actuators*)	None	Unstable	None	Unstable
Percentage of overshoot in pitch angle (truth model= .6 mach/20,000 feet)	100	35	50	10
Percent of initial condition present in oscillation at 2 seconds (truth model= .6 mach/20,000 feet)	None	5	15	10

* Truth model is otherwise identical to linear design model

Table E-5. Summary of Minimum Singular Values (Inverse Return Difference) and Regulator Pole Locations for New Sample Designs (Part 1)

Design	$\underline{\sigma}_{0.1}$	$\underline{\sigma}_{1.0}$	$\underline{\sigma}_{10}$	$\underline{\sigma}_{100}$	$\underline{\sigma}_{min}$	Poles
SR-1C	0.93	0.80	7.4	140	0.80	-19.9 -22.8 -35.4 -1.8 ± j0.06 -17.1 ± j21.3
SR-2C	0.95	0.94	15.8	250	0.92	-11.2 -20.0 -23.5 -2.2 ± j0.08 -10.9 ± j11.4
IMF2-1C	0.95	0.91	12.6	200	0.91	-2.66 -5.04 -25.7 -12.5 ± j12.9 -18.1 ± j0.07
IMF2-2C	0.95	0.92	11.2	200	0.90	-2.0 -2.4 -12.9 -18.97 -26.1 -12.6 ± j15.3

Table E.6 Summary of Minimum Singular Values (Inverse Return Difference) and Regulator Pole Locations for New Sample Designs (Part 2)

Design	$\underline{\sigma}_{0.1}$	$\underline{\sigma}_{1.0}$	$\underline{\sigma}_{10}$	$\underline{\sigma}_{100}$	$\underline{\sigma}_{min}$	Poles
IMF2-3C	0.98	0.95	15.8	250	0.92	-2.02 -2.39 -11.7 -19.7 -23.5 -11.0 ±j11.5
IMF3-1C	0.93	0.80	7.1	140	0.80	-5.6 -8.8 -15.9 -24.4 -18.1 ±j0.37 -20.7 ±j21.1
IMF3-2C	0.97	0.90	19.9	220	0.91	-2.67 -6.01 -11.2 -11.2 -19.3 -19.97 -10.4 ±j6.8
IMF4-1C	0.99	0.96	8.9	100	0.96	-15.9 -15.9 -19.7 -20.0 -5.13 ±j3.9 -254.4 ±j157.1

VITA

William Gilbert Miller was born on 1 May 1950 in Corry, Pennsylvania, and graduated from Corry Area High School in 1968. In 1972 he received the degree of Bachelor of Science, Summa Cum Laude, from Grove City College, and obtained a commission in the USAF through the ROTC program. He completed pilot training at Sheppard AFB, Texas, in August 1973. From then until September 1976, he was assigned to the 5041st Tactical Operations Squadron at Elmendorf AFB, Alaska, as a T-33 pilot, instructor pilot and flight examiner. From 1976 to 1980, he served first as a T-33 instructor pilot, and then as an F-106 pilot and Weapons Instructor with the 87th Fighter Interceptor Squadron at K. I. Sawyer AFB, Michigan. In January 1980 he was assigned to the USAF Interceptor Weapons School at Tyndall AFB, Florida, as an F-106 Weapons Instructor. In May 1982 he received the degree of Master of Science in Systems Management from the University of Southern California. In June 1982 he entered the School of Engineering, Air Force Institute of Technology.

Permanent address: c/o Dr. Allan Serviss
Box 418
Chester Springs, PA 19425

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/83D-48	2. GOVT ACCESSION NO. AD-A138 309	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ROBUST MULTIVARIABLE CONTROLLER DESIGN VIA IMPLICIT MODEL-FOLLOWING METHODS		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) William G. Miller Capt		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December, 1983
		13. NUMBER OF PAGES 349
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

7 Feb 84

Approved for public release: IAW AFR 190-17.
Lynn Wolaver
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Optimal Control
Model-Following Control
Proportional-Integral Control
Robustness
Singular Value Decomposition

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This study applies the concept of implicit model-following to the design of a controller based on the Linear system model, Quadratic cost, and Gaussian noise process (LQG) assumptions of optimal control. The objective is to achieve a controller design with a Proportional-plus-Integral (PI) regulator that will work well despite "uncertainties" arising from unmodelled higher-order dynamics, parameter variations, and nonlinearities. Designs are presented for the decoupled pitch-pointing control of a modern fighter aircraft (the AFTI F-16). The designs are achieved using software that resulted from previous AFIT thesis efforts. Design robustness is analyzed by means of matrix singular value analysis and actual simulation with realistic truth models; the specific analysis software used was developed for this study, and is documented in the report. Implicit model-following is discussed and shown to provide a means of achieving the stated design objective. Singular value analysis appeared to be of limited value in predicting the relative robustness of controller designs for the problem considered.

4-8